Escola Politècnica Superior
de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE CARRERA

TÍTULO DEL TFC: Desarrollo de un sistema de ayuda a la navegación para helicópteros

TITULACIÓN: Ingeniería Técnica Aeronáutica, especialidad Aeronavegación

AUTOR: Gonzalo Sainz de Baranda Garrido

DIRECTOR: José Manuel Sainz de Baranda Graf

FECHA: 30 de noviembre de 2010

Título: Desarrollo de un sistema de ayuda a la navegación para helicópteros

Autor: Gonzalo Sainz de Baranda Garrido

Director: José Manuel Sainz de Baranda Graf

Fecha: 30 de noviembre de 2010

Resum

El objetivo de este trabajo es el desarrollo de un sistema de ayuda a la navegación para helicópteros. Para ello partiendo de un visualizador de cartografía (IPSIGEMAP) desarrollado por RSB Sistema se incorporará una base de datos de obstáculos y se creará un control que permitirá visualizar la posición del helicóptero sobre la cartografía y emitirá posibles alarmas de colisión. En todo momento se buscará que el sistema sea totalmente independiente de los sistemas del helicóptero.

Para ello se tendrá que:

1. Aprender a programar en los lenguajes Javascript y HTML a través de los cuales se implementarán las nuevas funciones al visor.
2. Añadir la dimensión Z al visor cartográfico.
3. Crear una nueva clase de objetos geométricos en el visor.
4. Investigar y crear algoritmos de colisión de objetos geométricos.
5. Extraer y encontrar información a partir de receptores GPS, tales como la localización de un móvil, su velocidad horizontal, velocidad vertical, altitud o rumbo, todo ello con la utilización del protocolo NMEA, con objeto de obtener las variables de ubicación y desplazamiento en tiempo real del helicóptero.
6. Utilizar modelos digitales de elevación del terreno, MDE, y crear rutinas para gestionarlos.
7. Implementar utilidades de control y aviso de alarmas en el visor que nos permitan ver la posición del helicóptero y de los obstáculos en todo momento y nos avise en caso de peligro de colisión con estos o con el suelo.
8. Creación de un control de manipulación de la posición que simule un objeto helicóptero para el testeo de las nuevas características del visor cartográfico.

Title: Desarrollo de un sistema de ayuda a la navegación para helicópteros

Author: Gonzalo Sainz de Baranda Garrido

Director: José Manuel Sainz de Baranda Graf

Date: 30 de noviembre de 2010

Overview

The aim of this Project consists on developing an aid system for helicopters' navigation. In order to do this, starting with a cartography visual display unit (IPSIGEMAP) developed by RSB system , it will be incorporated an obstacle database and an established control which will be able to visualize the helicopter's position over the cartography giving out alarms warning about possible crashes. At all times, the objective will be that this system works regardless of helicopters' systems.

It requires:

1. To learn how to program in Javascript and HTML languages in which the new visor functions will be installed.
2. To add the Z dimension to the visor.
3. To create a new kind of visor's geometrical objects.
4. To investigate and create collision algorithms of geometrical objects.
5. To extract and find information from GPS receptors such as a vehicle finding , its horizontal speed, vertical speed , altitude or course using the NMEA protocol, with the aim of obtaining location and helicopter's real-time displacement variables.
6. To use digital elevation models and creating programs in order to facilitate it.
7. To establish control profits and alarms warnings in order to check the helicopter and obstacle's position all the time and also with the aim of been informed in cases of having a risk of collision.
8. The creation of a "position's control" which simulates an helicopter object in order to test the new visor's characteristics.

ÍNDICE

CAPÍTULO 1. INTRODUCCIÓN 1

1.1. Introducción al IPSIGEMAP 1

1.2. Introducción a la cartografía 2
1.2.1. ELIPSOIDE 2
1.2.2. GEOIDE 3
1.2.3. DATUM 4
1.2.3.1. Datum ED50 4
1.2.3.2. ETRS 89 5
1.2.4. Sistemas de coordenadas 5
1.2.4.1. Proyecciones 5
1.2.4.1.1. Proyecciones geodésicas 6
1.2.4.1.2. Proyección UTM 6
1.2.4.1.3. Proyección conforme de Lambert 8
1.2.4.2. Coordenadas Geográficas 8
1.2.4.2.1. Coordenadas geográficas no proyectadas 8
1.2.4.2.2. Coordenadas geográficas proyectadas 9
1.2.4.3. Coordenadas Geocéntricas. 9
1.2.5. GPS 10
1.2.5.1. Introducción 10
1.2.5.2. Protocolo NMEA 10

CAPÍTULO 2. ARQUITECTURA DEL SISTEMA IPSIGEMAP 12

2.1. Esquema básico y componentes 12

2.2. Organización de la información geográfica 13
2.2.1. Capa base 13
2.2.2. Superposiciones externas 13
2.2.3. Superposiciones internas 13

2.3. Motor cartográfico del sistema ipSigeMap (IPSM) 14
2.3.1. Fuentes cartográficas soportadas 14
2.3.2. Funcionalidades del motor 15

2.4. Traductores y el componente IPSMGM 16

2.5. Interfaz de usuario y funciones de alto nivel (geoIpSige) 16

2.6. MapServices 18

CAPÍTULO 3. SOFTWARE DE OBTENCIÓN Y TRATAMIENTO DE DATOS 20

3.1. La nomenclatura EPSG 20

3.2. Cambio Coordenadas 20

3.3. Altura del terreno 21

3.4. Datos de navegación 23

3.4.1. sentencia $GPGSA 23
3.4.2. sentencia $GPGGA 24
3.4.3. sentencia $GPRMC 24

3.5. Tratamiento de datos NMEA 25

CAPÍTULO 4. SOFTWARE DE ANÁLISIS DE LAS COLISIONES GEOMÉTRICAS 26

4.1. Intersección de paralepípedos envolventes 26

4.2. Análisis de la colisión paralepípedo-polilínea 27
4.2.1. Getlimites 28
4.2.2. Determinación de la posible colisión 29

4.3. Análisis de la colisión paralepipedo- paralepipedo 30
4.3.1. Getlimites 30
4.3.2. Determinación de la posible colisión 30

4.4. Análisis de la colisión paralepípedo-cilindro 31
4.4.1. Getlimites 32
4.4.2. Calculo intersección subapartado 32

CAPÍTULO 5. SOFTWARE DE AMPLIACIÓN DE LA INTERFAZ DE USUARIO DEL VISOR 38

5.1. Visualización de las coordenadas 38

5.2. Control del helicóptero 39

5.3. Visualización de obstáculos 39

5.4. Control de posición 41

CAPÍTULO 6. CONCLUSIONES 42

REFERENCIAS 43

ANEXO 1. FORMULAS DE CAMBIOS DE COORDENAS 45

ANEXO 2. MODELO DIGITAL DE ELEVACIONES 62

ANEXO 3. GESTIÓN DEL GPS 64

3.1. Gestión de las tramas NMEA 64

3.2. Servlet de intercomunicación de la posición actual 68

ANEXO 4. AMPLIACIÓN DE LA INTERFAZ DE USUARIO DEL VISOR 73

4.1. Visualización de las coordenadas 73

4.2. Control helicóptero 73

4.3. Control de la posición..... 76

ÍNDICE DE FIGURAS

Fig. 1.1 Imagen del visor cartográfico IpSigeMap.....1

Fig. 1.2 Comparación entre el Geoide y el Elipsoide.....4

Fig. 1.3 Proyección UTM (Universal Transverse Mercator).....6

Fig. 1.4 Proyección Mercator.....7

Fig. 1.5 Proyección conforme de Lambert.....8

Fig. 1.6 Coordenadas terrestres egocéntricas.....9

Fig. 1.7 Intersección de tres satélites GPS.....10

Fig. 2.1 Esquema de bloques del sistema IpSigeMap.....12

Fig. 2.2 Imagen del visor IpSigeMap donde se observan sus diferentes niveles.....14

Fig. 3.1 Gráfico cambio de coordenadas.....21

Fig. 3.2 Diferencia de altura entre el geoide y el elipsoide.....22

Fig. 4.1 Error común en paralepípedos alargados.....32

Fig. 4.2 Distancia a la recta de un segmento desde un punto.....33

Fig. 5.1 Vista de la Escola Politècnica Superior de Castelldefels desde el visor IpSigeMap.....38

Fig. 5.2 “control del helicóptero” en el visor IpSigeMap.....39

Fig. 5.3 Alarma de posible colisión.....40

Fig. 5.4 Control de posición en el visor IpSigeMap.....41

CAPÍTULO 1. INTRODUCCIÓN

1.1. Introducción al IPSIGEMAP

El SIGE (Sistema de Información para la Gestión de Emergencias), es un programa creado, como su propio nombre indica, para la gestión de emergencias, fundamentalmente por los cuerpos de bomberos; permite gestionar recursos, dotaciones, turnos,...., así como las propias emergencias, indicando donde se ubican, proponiendo los adecuados protocolos de intervención (que vehículos, personal y herramientas se recomiendan enviar) así como sugerencias sobre el camino mas rápido o la manera de intervenir.
En los últimos años se ha migrado a tecnologías basadas en la utilización de la "web" añadiendo el prefijo ip al nombre (ipSIGE).

El Sistema de Información para la Gestión de Emergencias dispone de un visualizador de cartografía, ipSigeMap, que permite la ubicación espacial de los elementos con los que se trabaja. Este visualizador permite la visualización de los principales formatos de fuentes cartográficas, tanto vectoriales como raster.

El objetivo último del sistema ipSigeMap es disponer de una familia de componentes que se puedan cargar directamente en un navegador web o incrustar en una página web y permitan la visualización de cartografía y de la ubicación espacial de los elementos gestionados por el SIGE (o por cualquier otro sistema) así como ofrecer al usuario una serie de funcionalidades avanzadas de gestión cartográfica y georreferenciación.

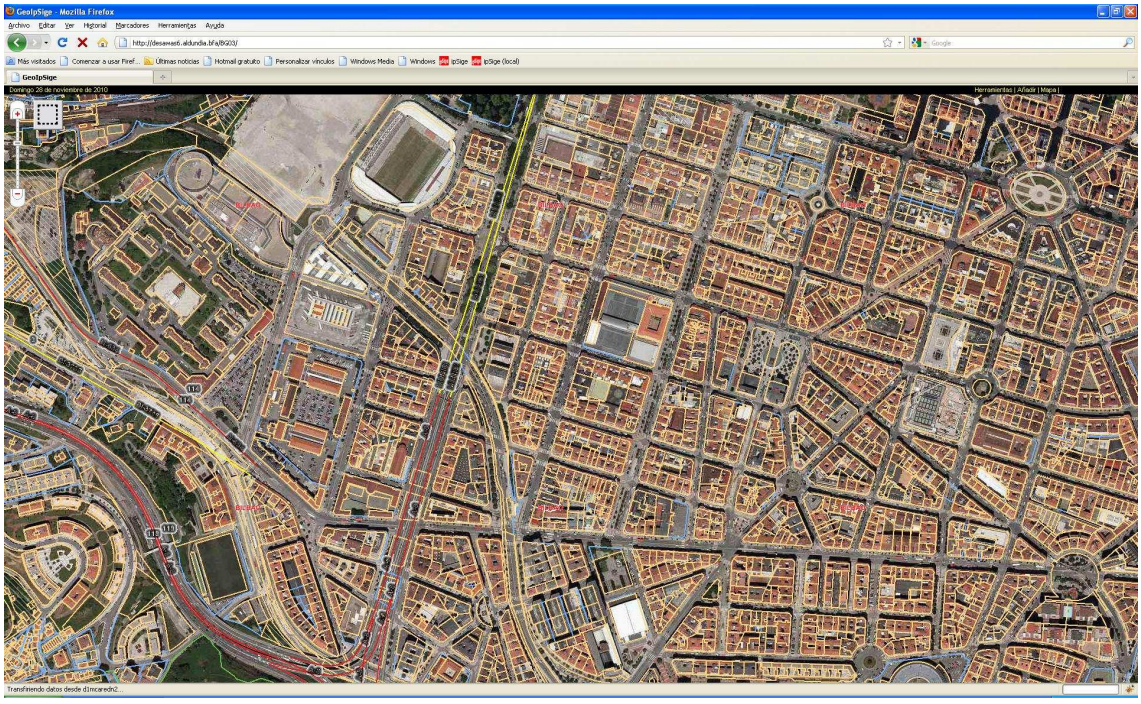

Fig. 1.1 Imagen del visor cartográfico IpSigeMap

1.2. Introducción a la cartografía

La superficie de la Tierra no es plana, se tiende a decir que es una esfera, lo cual para cálculos sencillos y aproximados es práctico, pero en realidad es más complejo, achatada en los polos y abultada en el ecuador, más próxima a una pera.

Varios pensadores griegos (Pitágoras, Aristóteles, Arquímedes y Platón) ya se aventuraron a defender que la Tierra era una esfera, si bien mas por razones religiosas ya que consideraban a la esfera el sólido mas perfecto.

La primera persona en estimar la circunferencia de la Tierra fue el filósofo griego Eratóstenes de Cierne (276 AC al 194 AC), al cual habían informado que en el pueblo de Syene (llamado Aswan en el Egipto moderno), el día del solsticio de verano, a mediodía, el sol se reflejaba en las aguas de un pozo. No obstante el había observado que el mismo día en Alejandría el sol no estaba totalmente vertical, así que midió el ángulo de la vertical con la sombra que generaba un obelisco a mediodía (7º 12') y contrató a un contador de pasos para que estimara la distancia entre Alejandría y Syene. Estimó el radio de la Tierra en 6267 Km, solo un 2% de error con el aceptado hoy en día. Para ello desarrolló un sistema que ya contenía los conceptos de latitud y longitud.

Newton en 1687 propuso como mejor aproximación de la forma de la tierra al elipsoide de revolución, modelo que fue perfeccionado posteriormente por Cassini con la formulación práctica que dura hasta nuestro días. Hasta 1924, cada Nación utilizaba el elipsoide que mejor se adaptaba localmente a su superficie, y así la geodesia española se refirió al elipsoide de Struve. En 1924, se adoptó para todo el mundo como superficie de referencia el elipsoide de Hayford.

Actualmente podría decirse que un mapa se construye considerando el elipsoide, el tipo de proyección y el datum utilizados.

1.2.1. ELIPSOIDE

En general se considera a la Tierra como un elipsoide de revolución alargado según la línea de los polos, con una excentricidad dada por:

$$e^2 = (b^2 - a^2) / a^2 \qquad \textbf{(1.1)}$$

o un achatamiento

$$f = (a - b) / a \qquad \textbf{(1.2)}$$

Forma que se asemeja a la forma de la Tierra al utilizar las coordenadas geográficas y es una figura matemática fácil de usar. Existen varios elipsoides

de referencias, pero el mas utilizado hoy en día es el World Geodetic System 84 (WGS-84), desarrollado por el departamento de defensa de EEUU y utilizado por el sistema de posicionamiento GPS.

Tabla 1.1 Parámetros de elipsoides de referencia

Nombre	a (m)	b (m)	$1/f$
Australian National	6378160.000	6356774.719	298.250000
Bessel 1841	6377397.155	6356078.963	299.152813
Clarke 1866	6378206.400	6356583.800	294.978698
Clarke 1880	6378249.145	6356514.870	293.465000
Everest 1956	6377301.243	6356100.228	300.801700
Fischer 1968	6378150.000	6356768.337	298.300000
GRS 1980	6378137.000	6356752.314	298.257222
International 1924 (Hayford)	6378388.000	6356911.946	297.000000
SGS 85	6378136.000	6356751.302	298.257000
South American 1969	6378160.000	6356774.719	298.250000
WGS 72	6378135.000	6356750.520	298.260000
WGS 84	6378137.000	6356752.314	298.257224

1.2.2. GEOIDE

A la hora de medir altitudes, el elipsoide no es la forma adecuada, para ello se introduce el geoide tomando como referencia el nivel del mar, ya que éste es alterado por los campos gravitacionales de la Tierra, este geoide será definido como: la superficie equipotencial del campo gravitatorio de la Tierra que mejor se ajusta (en el sentido de mínimos cuadrados), al nivel medio global del mar, es decir, el geoide es siempre perpendicular al vector de gravedad local en cada punto.

Se han desarrollado diversos modelos matemáticos del geoide.

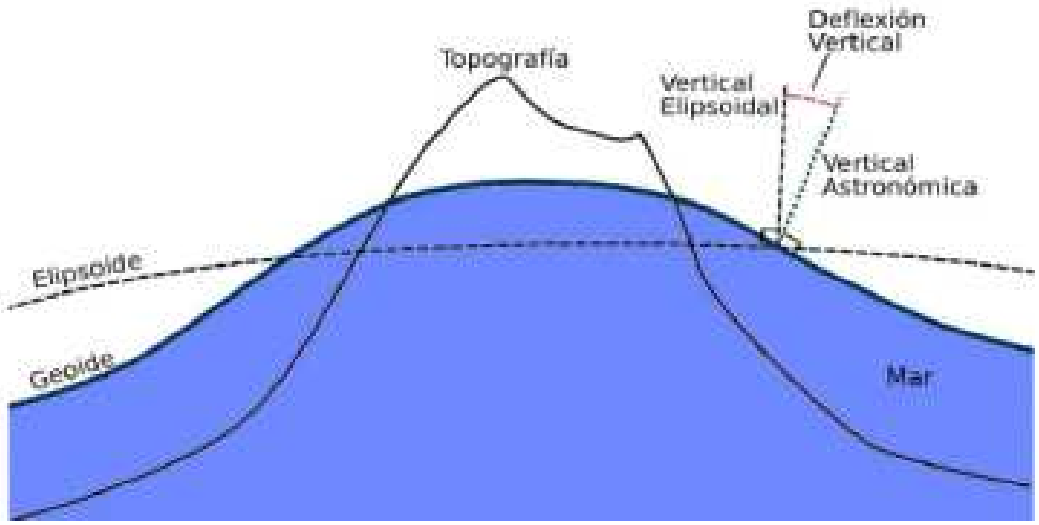

Fig. 1.2 Comparación entre el Geoide y el Elipsoide

1.2.3. DATUM

Como ya hemos visto es posible aproximar la forma de la Tierra con un elipsoide o un geoide , no obstante esto no es del todo preciso a la hora de calcular las alturas. Para solucionarlo se construyen elipsoides locales que minimicen las diferencias entre la altura del geoide o elipsoide global con dicho elipsoide local, esto es lo que llamamos datum.

Para construir un datum lo primero que necesitaremos es un elipsoide de referencia y un lugar preciso utilizado como referencia para definir el resto de puntos del mapa (Potsdam en el caso del ED:50), este lugar será llamado punto de referencia.

Una vez definidos el resto de los puntos, se pueden considerar diferentes tipos de proyecciones de la superficie curva de la Tierra en un mapa plano.

1.2.3.1. Datum ED50

El datum ED50 (European Datum 1950) es un antiguo sistema de referencia geodésico empleado en Europa. En España ha sido el sistema oficial para la cartografía de la Península y Baleares hasta 2008, año en que fue sustituido por el ETRS89 (explicado mas abajo).

ED50 surgió como el resultado de la unificación, por parte del ejército de los Estados Unidos, de los antiguos sistemas europeos después de la segunda guerra mundial.

El elipsoide que utiliza es el internacional de 1924 o de Hayford cuyas características de longitud y achatamiento están explicadas en la tabla 1.1. El punto de origen o astronómico fundamental está situado en la torre de Helmert, en la ciudad alemana de Postdam. Otros datum de posterior desarrollo como el

ED79 o el ED87 tomarían como punto de origen la ciudad también alemana de Munich pero estos datum no pasaron de especulaciones científicas y no se ha llegado a realizar cartografía oficial con ellos. En el caso de España para la altura se toma como punto de referencia el nivel medio del mar en Alicante.

Su código EPSG correspondiente es EPSG:4230

1.2.3.2. ETRS 89

El ETRS89 (European Terrestrial Reference System 1989) es el sistema de referencia geodésico utilizado hoy en día en Europa. La razón por la cual se adopta es que es más preciso que el anterior ED50 al tener en cuenta el movimiento de la placa continental europea (3cm por año).

Está basado en el elipsoide GRS80 cuyas características de longitud y achatamiento están explicadas en la tabla 1.1.

Su código EPSG correspondiente es EPSG:4258.

1.2.4. Sistemas de coordenadas

Tanto para la proyección cartográfica de la superficie terrestre, como para indicar posiciones sobre ella es necesario establecer un modelo de su superficie que nos permita fijar un sistema de coordenadas. El modelo más simple utilizado para la Tierra es una esfera. Las coordenadas que utilizamos pueden ser proyectadas, geográficas o geocéntricas.

1.2.4.1. Proyecciones

Un problema que han tenido los cartógrafos desde su origen era como representar una superficie esférica sobre un mapa plano. Para ello se desarrollaron un amplio número de proyecciones según el conjunto de reglas que se escojan, cada uno de estos conjuntos de reglas define diferentes propiedades e introducen distorsiones inevitables, la razón de que existan tantos tipos diferentes de proyecciones es que se adecuan para el lugar o el uso que se desee.

Algunas de estas propiedades son:

- Conformidad: Un mapa conforme es aquel que preserva los ángulos (y por tanto, las formas) a nivel local.
- Equivalencia: Una proyección es equivalente o autálica si mantiene las proporciones entre las áreas representadas.
- Equidistancia: Se dice que una proyección es equidistante cuando posee un conjunto bien definido y completo de líneas a lo largo de las cuales la escala se mantiene constante.
- Dirección: Si distorsionan o no las direcciones.
- Tipo de superficie y orientación: Si es sobre un plano o una superficie y su orientación respecto al plano del ecuador.

- Posición de la superficie de proyección: Si son tangentes o “cortan” a la Tierra.
- Posición del punto de proyección: Si es el centro de la Tierra, un punto de la Superficie o un punto fuera de esta.

1.2.4.1.1. Proyecciones geodésicas

Las proyecciones geodésicas son aquellas en las que la esfericidad terrestre se tiene en cuenta a la hora de representar posiciones geográficas, ángulos, distancias y superficies. Dos de las proyecciones geodésicas más utilizadas son la UTM y la conforme de Lambert.

1.2.4.1.2. Proyección UTM

El sistema de coordenadas UTM (inglés Universal Transverse Mercator), fue creado por el cuerpo de ingenieros del ejercito de los estados unidos en la década de los 40. Como su propio nombre indica, se basa en la proyección transversal de Mercator que fue creada por el geógrafo flamenco Gerardus Mercator en 1659, proyectando la superficie de la tierra sobre un cilindro tangente al ecuador que posteriormente se despliega.

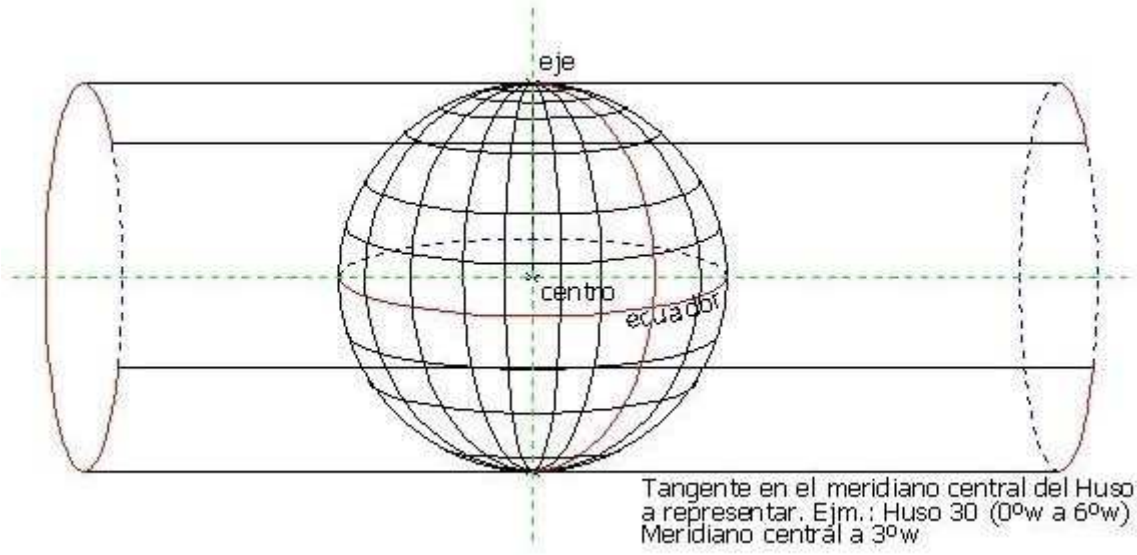

Fig. 1.3 Proyección UTM (Universal Transverse Mercator)

Esta proyección es conforme, es decir mantiene los ángulos casi no distorsionando las formas pero sacrificando las distancias y áreas.

La proyección UTM se construye igual con la diferencia que en vez de hacerla tangente al ecuador se hace a un meridiano. Para ello utiliza un elipsoide de la Tierra. En sus inicios se utilizo el Internacional, excepto para el área de Estados Unidos para la cual se utilizo el elipsoide de Clarke (1866),

actualmente se utiliza el elipsoide ETRS 89 en Europa como modelo base, al tener en cuenta, como se comenta mas arriba, el movimiento de las placas tectónicas.

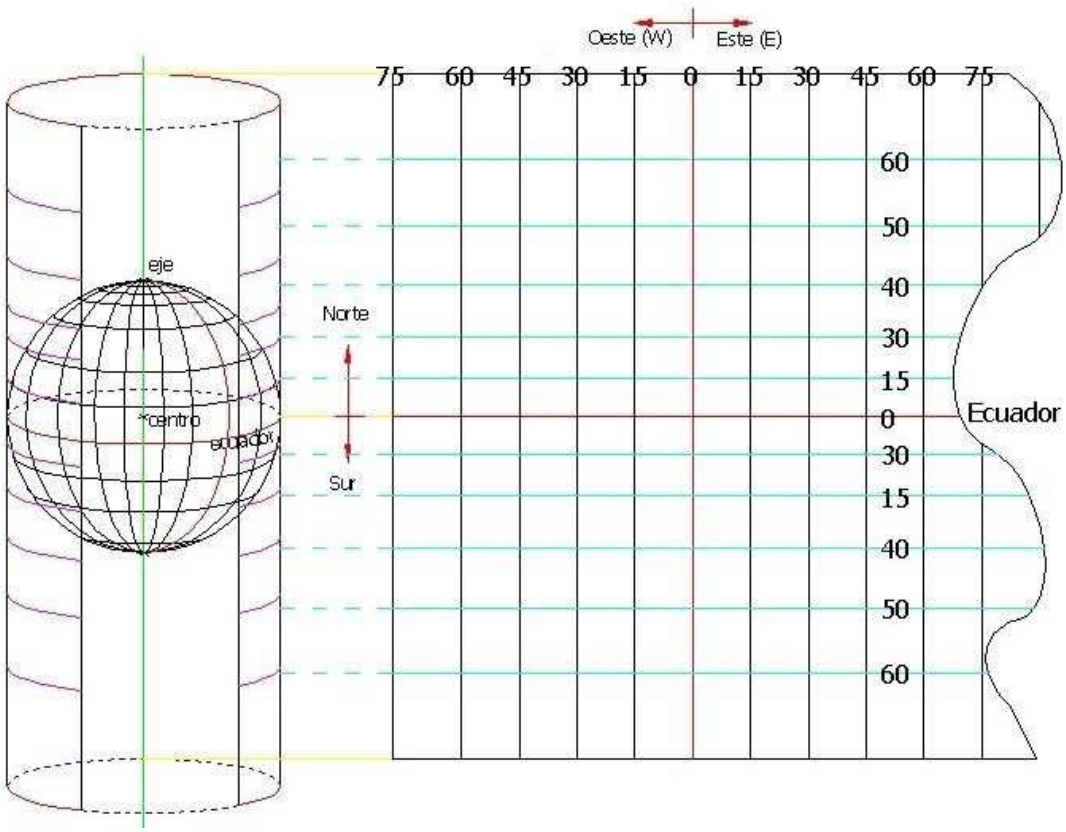

Fig. 1.4 Proyección Mercator

Son coordenadas proyectadas, es decir, es un sistema de representación gráfico que establece una relación ordenada entre los puntos de la superficie curva de la Tierra y los de una superficie plana (mapa) en forma de cuadriculas “locales”.

Es importante tener en cuenta que según el meridiano que se utilice se obtienen distintas proyecciones. Se ha dividido la tierra en 60 gajos de 6º cada uno denominados husos. En la península Ibérica se utilizan los husos 29, 30 (meridiano de Greenwich) y 31.

Anteriormente al desarrollo del sistema UTM, en el periodo de entreguerras, varios países europeos ya habían experimentado la utilidad de mapas cuadriculados, en proyección conforme, viendo que el calculo de distancias entre dos puntos en estos mapas era más sencillo aplicando el teorema de Pitágoras que las fórmulas trigonométricas de los mapas referenciados en longitud y latitud. Estos conceptos acabarían extendiéndose al sistema UTM y a la Estereográfica Polar Universal, que es un sistema cartográfico mundial basado en cuadricula recta.

1.2.4.1.3. Proyección conforme de Lambert

La proyección conforme de Lambert es una proyección cartográfica cónica muy utilizada en navegación aérea, la cual no debe confundirse con la azimutal de Lambert.

Es una proyección cónica, por lo tanto superpone un cono sobre la superficie de la Tierra y lo despliega, con 2 paralelos de referencia secantes al globo, esto minimiza la distorsión al proyectar de tres dimensiones a dos, siendo mínima a lo largo de los paralelos de referencia, pero aumentando a medida que se aleja de ellos. Como su nombre indica es conforme.

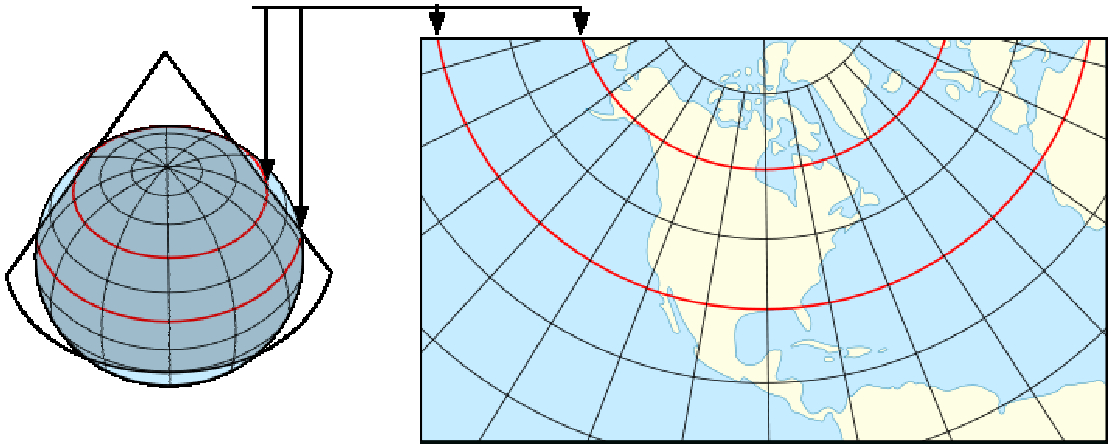

Fig. 1.5 Proyección conforme de Lambert

Lambert ajustó matemáticamente la distancia entre paralelos para crear un mapa conforme, como los meridianos son líneas rectas y los paralelos arcos de circulo concéntricos, las diferentes hojas encajan perfectamente.

Los pilotos utilizan estas cartas ya que una línea recta mostrara la distancia verdadera entre dos puntos, no obstante la ruta deberá ser calculada independientemente pues será una línea curva referente al círculo máximo entre esos 2 puntos.

1.2.4.2. Coordenadas Geográficas

1.2.4.2.1. Coordenadas geográficas no proyectadas

El sistema consiste en dividir la superficie en una cuadricula bidimensional definida por los parámetros longitud y latitud.

La longitud mide el ángulo a lo largo del ecuador desde cualquier punto de la Tierra, es decir a cuanto se halla hacia el este o el oeste, se toma como el

punto de referencia 0 Greenwich, las líneas de longitud se llaman meridianos y son círculos máximos que pasan por los polos.

La latitud por su parte mide el ángulo entre un punto y el ecuador, es decir, cuanto se encuentra hacia el norte o hacia el sur, las líneas de latitud se llaman paralelos y son paralelas al ecuador de la Tierra.

1.2.4.2.2. Coordenadas geográficas proyectadas

El desarrollo de sistemas cartográficos digitales ha implicado la utilización de proyecciones directamente de las coordenadas geográficas, debido a la necesidad de un único sistema de proyección válido para toda la superficie de la tierra. Uno de los ejemplos más populares en la aplicación Maps de Google, que usa la proyección de Mercator. A pesar de sus relativas distorsiones de escala, esta proyección es bastante indicada para un mapa interactivo en que se hacen desplazamientos y zooms en regiones pequeñas, donde las formas se distorsionan relativamente poco. En los mapas en Google Maps la máxima latitud es +/- 85.0511287798066 grados.

1.2.4.3. Coordenadas Geocéntricas.

Las coordenadas geocéntricas tienen su origen en el centro de la Tierra y pueden ser coordenadas cartesianas (x, y, z), coordenadas eclípticas (angulares) o coordenadas ecuatoriales (ascensión, declinación).
Por ejemplo el elipsoide WSG-84, elipsoide de referencia de la constelación GPS, es un elipsoide que toma como origen el centro de gravedad de la Tierra.

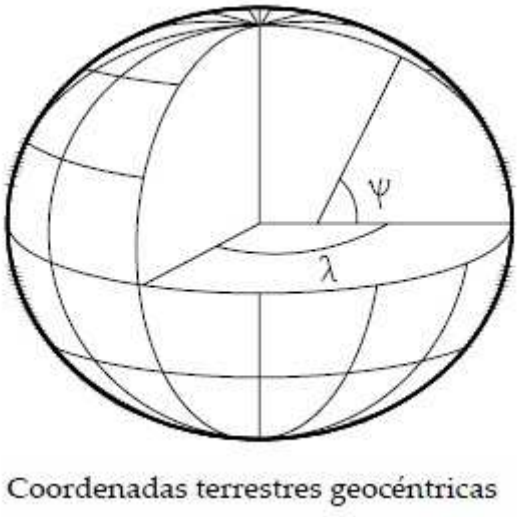

Fig. 1.6 Coordenadas terrestres geocéntricas

1.2.5. GPS

1.2.5.1. Introducción

GPS es un Sistema Global de Navegación por Satélite (GNSS, en su acrónimo en ingles), aunque existen otros sistemas como el Galileo Europeo (en fase de despliegue) o el GLONASS ruso; hoy en día, GPS es el más utilizado.

Consta de 3 segmentos:

- Segmento espacio: Formado por una constelación de 24 satélites repartidos en 6 órbitas.
- Segmento control: Estaciones en tierra que permiten gestionar y realizar el mantenimiento de las constelaciones
- Segmento usuario: Receptor que indica su situación.

Su funcionamiento se basa en calcular el tiempo transcurrido en recibir una señal, esto nos indica la distancia desde el emisor (satélite) y receptor, para ello son necesarios 3 satélites.

Cada satélite nos indica que el receptor se encuentra en un punto de la superficie de una esfera con centro en el satélite y radio igual a la distancia entre este y el receptor.

Con la unión de estas 3 esferas nos salen 2 puntos posibles de situación, pudiendo descartar el que está dentro de la corteza terrestre por razones obvias. Debido a que los relojes de los receptores no están sincronizados con los de los satélites, es necesaria la utilización de un 4 satélite para conocer una situación correcta en 3D.

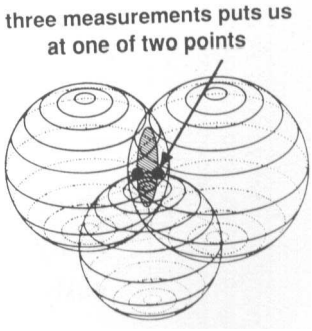

Fig. 1.7 Intersección de tres satélites GPS

1.2.5.2. Protocolo NMEA

Para transmitir los datos GPS existen varios protocolos, en este caso se utiliza el protocolo NMEA (National Marine Electronic Asociation) asociación precursora de la comunicación entre equipos de navegación, la cual creó este

protocolo para comunicar al piloto automático de los barcos, su desviación respecto a una trayectoria predeterminada y las coordenadas de su posición.

El protocolo NMEA envía los bloques de datos en líneas denominadas sentencias, cada elemento de la línea tiene su significado y se ha de analizar por separado. Todas las sentencias comienzan con un `$`, seguido de dos letras que nos indican que tipo de dispositivo es (GP para GPS o LC para Loran) y finaliza el primer elemento de la línea con tres letras que indican que datos son el resto de la línea, como indica el siguiente ejemplo:

$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47

$GPGGA:	Tipo de sentencia. Global Positioning System Fix Data
123519:	Hora GPS. 12:35:19
4807.038,N:	Latitud. 48º 07.038’ Norte
01131.000,E	Longitud. 11º 31.000’ Este
1:	Calidad de la señal. 0 = Inválida, 1 = GPS, 2 = DGPS
08:	Número de satélites captados
0.9:	HDOP. Referencia del nivel de precisión horizontal
545.4,M:	Altura sobre el nivel del mar. 545.4 metros
46.9,M:	Altura medida en geoide sobre el elipsoide modelo WGS84
*47:	Número de chequeo de errores

Para este proyecto utilizamos las sentencias tipo GPGSA, GPGGA y GPRMC las cuales explicaremos mas adelante.

CAPÍTULO 2. ARQUITECTURA DEL SISTEMA IPSIGEMAP

2.1. Esquema básico y componentes

El esquema de bloques básico del sistema IpSigeMap es el siguiente:

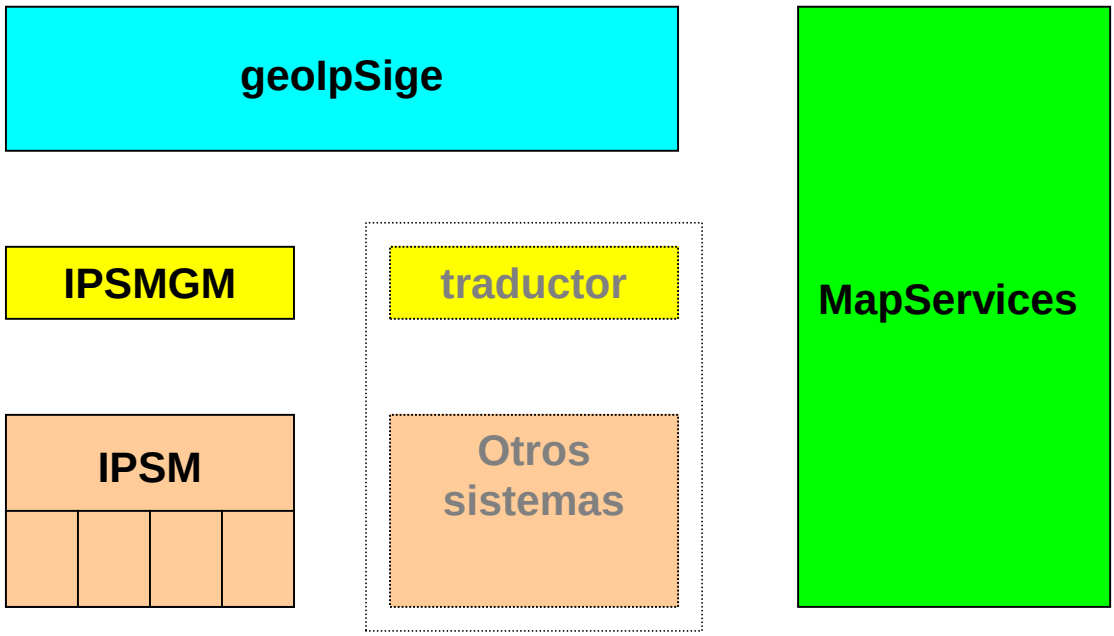

Fig. 2.1 Esquema de bloques del sistema IpSigeMap

Los componentes sobre los que interactúa el usuario son los superiores, en concreto el módulo geoIpSige que convierte las acciones del usuario en llamadas a clases y procedimientos abstractos que luego son implementados por el motor del sistema (IPSM).

Los bloques incluidos en la caja punteada son los que podrán sustituir al motor del sistema (IPSM) en aras de la compatibilidad (google maps por ejemplo), cada motor utilizado necesitará un traductor que convertirá el esquema real de clases del motor utilizado en el esquema de clases abstracto utilizado por geoIpSige.

Como se puede observar en la parte inferior el motor del sistema ipSigeMap (IPSM) dispone de diversos módulos para poder visualizar diversas fuentes cartográficas.

El componente geoIpSige es el responsable de la gestión de la interfaz de usuario y el soporte de las funciones de alto nivel.

En el componente MapServices se incluyen un conjunto de servicios web para el apoyo a la georreferenciación, acceso a información alfanumérica, obtención de ficheros para gestión de superposiciones internas, interconexión con el SIGE, etc.

2.2. Organización de la información geográfica

Antes de analizar las funcionalidades de cada componente es importante describir el procedimiento de organización de la información cartográfica que será visualizada por el usuario.

La información se organizará en tres niveles:

a) Capa base
b) Superposiciones externas
c) Superposiciones internas

2.2.1. Capa base

La capa base será la capa fondo sobre la que se dibujarán el resto de los elementos.

Normalmente se construirá con una fuente cartografía raster, como las ortofotos, o con una fuente vectorial rasterizada aunque teóricamente puede ser cualquier tipo de fuente. Será necesario disponer de imágenes opacas para cada nivel de zoom.

El sistema está preparado para poder elegir entre diversas capas base.

2.2.2. Superposiciones externas

Serán todas las fuentes cartográficas que se puedan dibujar sobre la capa base y que residan en distintos servidores externos. Normalmente serán fuentes vectoriales para que el servidor facilite imágenes transparentes (usualmente en formatos png o gif).

2.2.3. Superposiciones internas

En este apartado se incluyen todos los elementos que se dibujarán sobre la cartografía desde las aplicaciones cliente. Por ejemplo:

- Etiquetas
- Marcadores
- Polilíneas
- Polígonos

Estos elementos pueden crearse en respuesta a acciones del usuario o a partir de ficheros leídos en un servidor (ficheros xml, kml, shp, etc) o a partir de los servicios implementados en el módulo MapServices (por ejemplo representar la ubicación de los siniestros actuales)

En la siguiente imagen pueden observarse los tres tipos de niveles de información.

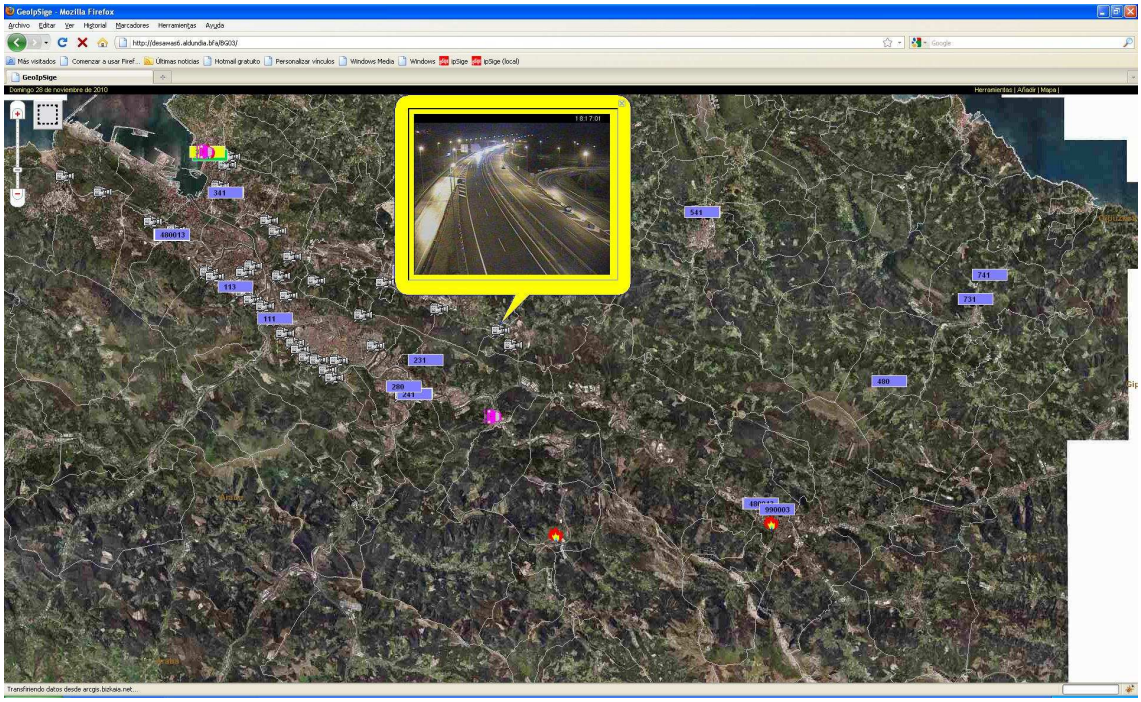

Fig. 2.2 Imagen del visor IpSigeMap donde se observan sus diferentes niveles

2.3. Motor cartográfico del sistema ipSigeMap (IPSM)

El componente IPSM es el responsable de la adecuada visualización y gestión de las fuentes cartográficas y del soporte de las funcionalidades del sistema, por lo que constituye el motor del ipSigeMap.

Consistirá en una librería escrita en javascript con un conjunto de sublibrerías para el tratamiento de las diversas fuentes cartográficas.

2.3.1. Fuentes cartográficas soportadas

En la versión actual del motor se incluyen los módulos para la gestión de las siguientes fuentes de cartografía:

- Acceso directo a ortofotos o a cualquier otra cartografía rasterizada que se construya en una pirámide de mosaicos de imágenes.
- Acceso a fuentes de cartografía vectorial (servicios de ArcIMS)
- Acceso a Map Services (WMS) públicos (también se pueden configurar servicios privados para poder visualizar otros tipos de fuentes cartográficas).
- Acceso a la cartografía de Google Maps

2.3.2. Funcionalidades del motor

En la versión actual del motor IPSM se incluyen las siguientes funcionalidades:

a) Gestión de la capa base

Se incluyen todas las funciones necesarias para gestionar las posibles capas base. Hay que tener presente que el sistema no limita a priori el tipo de capa base a visualizar. Mediante la adecuada configuración se puede visualizar cualquiera independientemente del sistema de proyección elegido y del sistema de coordenadas que se utilice.

En concreto se dispondrá de las siguientes funciones:

- Configuración (definición de la proyección, número de niveles de zoom, tamaño de cada imagen del mosaico, vista inicial, etc).
- Control dinámico de imágenes a cargar.
- Desplazamiento (incluyendo un mecanismo de carga predictiva de imágenes para aumentar la velocidad de visualización).
- Zoom con dos posibilidades, una con cambio del nivel del mosaico de imágenes utilizadas y otra con forzado de la resolución de las imágenes, eligiendo el sistema de forma automática en cada caso que tipo elegir. Al usuario se le presentarán diversas herramientas de gestión del nivel de zoom incluyendo una para seleccionar directamente la zona a visualizar (zoom ventana).
- Control centralizado de eventos.
- Procedimientos de conversión de coordenadas.

b) Gestión de superposiciones internas

Se soportarán las siguientes superposiciones internas:

- Etiquetas
- Marcadores (con contenido estático o dinámico)
- Polilíneas
- Polígonos

Las funcionalidades incluidas son:

- Organización de los elementos en capas o niveles.
- Añadir o eliminar superposiciones.
- Dibujo de las superposiciones (mediante objetos html o iconos).
- Movimiento de los elementos ligado al movimiento de la capa base.
- Arrastre de marcadores.

c) Gestión de ventanas de información

Asociadas a los elementos visualizados se pueden abrir ventanas que amplíen la información. Estas ventanas pueden contener información en formato html para contenidos estáticos o están asociadas a una url para mostrar contenidos dinámicos.

2.4. Traductores y el componente IPSMGM

Todas las funciones de alto nivel del componente geoIpSige se construyen mediante una familia de clases abstractas que no tienen implementación ("interface"). Para cada posible motor de gestión de la cartografía habrá que construir una implementación de estas clases abstractas sobre los componentes del motor.

En concreto, el componente IPSMGM incluye la implementación de estas clases para el motor cartográfico del sistema ipSigeMap (IPSM).

2.5. Interfaz de usuario y funciones de alto nivel (geoIpSige)

El módulo geoIpSige incluye todas las funciones de alto nivel del sistema ipSigeMap y facilita la interfaz de usuario.

Las funcionalidades que se incluirán en la versión actual son:

a) Gestión de la interfaz de usuario

Este componente construye un menú en el que se le ofrece al usuario el acceso a las funciones del sistema. En concreto, se incluyen las funciones siguientes:

- Acceso a las herramientas de control de visualización del motor cartográfico (zoom, desplazamiento, etc).
- Herramienta para visualización dinámica de coordenadas (en el sistema nativo del motor y en cualquier otro sistema elegido en la configuración).
- Herramienta para búsquedas georreferenciadas (ver detalle más adelante).
- Herramienta para gestión de rutas (ver más adelante).
- Herramienta para el dibujo de elementos (si hay capas que permitan la creación de elementos).
- Gestión de capas a visualizar.
- Configuración de los parámetros de visualización de cada capa.
- Selector de capa base que se desea utilizar.

b) Gestión de capas

El motor se encarga del dibujo de las superposiciones internas, pero este componente se encarga de su gestión y organización en capas.

También en este componente se configura como se visualizarán los componentes de cada capa incluyendo la visualización selectiva y diferenciada en función de condiciones.

Se incluyen las siguientes funciones:

- Control de la visualización de capas.
- Control del redibujado de capas dinámicas (por ejemplo seguimiento de vehículos).
- Configuración de la visualización.
- Control del contenido de las capas: incluye diversos procedimientos para obtener el contenido de cada capa aunque el más usual es solicitarlo a un servicio web que devolverá un fichero en formato xml o kml.

c) Cargador ajax para facilitar el acceso a los servicios web o a los servicios de ArcIms

d) Búsquedas georreferenciadas

El usuario tiene acceso a una herramienta que le facilita la búsqueda de direcciones, topónimos, etc. La búsqueda es lo más amplia posible y de acceso unificado. Escribiendo un texto se busca a la vez su existencia como dirección o su existencia como un elemento gestionado por el SIGE (por ejemplo un hidrante). En el caso de existir varios elementos que satisfagan la condición impuesta se muestran todos en una lista para que el usuario elija el que desee. Los elementos localizados se muestran con una etiqueta sobre la cartografía.

Se dispondrá de tres procedimientos de búsqueda (utilizables según el orden de exposición):

i) Caché interna: el sistema leerá un fichero de direcciones muy usuales (en xml) y lo mantendrá siempre en memoria para un primer nivel de búsqueda rápida.
ii) Acceso mediante el cargador ajax a un servicio web que devolverá los elementos encontrados en un fichero en formato xml o kml.
iii) Acceso a sistemas públicos de georreferenciación como Cartociudad o Google Maps

e) Búsqueda de rutas

Este componente incluye procedimientos para la gestión de rutas encontradas entre dos puntos incluyendo su visualización y análisis pero no incluirá ninguna función de alto nivel para la búsqueda de rutas. Esta se delegará a las funciones nativas de búsqueda de rutas del motor cartográfico. El motor del ipSigeMap utiliza en su versión actual el

componente de búsqueda de rutas de Google Maps para resolver este tema.

2.6. MapServices

Realmente el componente MapServices no es tal, sino que esta formado por un conglomerado de servicios web que tienen diversas tareas tales como la georreferenciación, la búsqueda de elementos, la interconexión con el SIGE, la creación de ficheros con el contenido de las capas, etc... Estos servicios pueden ser utilizados por el componente geoIpSige o al tener una interfaz pública pueden utilizarse desde cualquier otro sistema. Por ejemplo, el SIGE utiliza estos servicios para la localización de direcciones.

En la versión actual del proyecto ipSigeMap se incluyen los siguientes servicios (agrupados por utilidad):

a) Interconexión con el SIGE

Con las funciones siguientes:

- Lista georreferenciada de avisos pendientes.
- Lista georreferenciada de servicios abiertos.
- Desplazamiento de un aviso o un servicio (al desplazar la etiqueta correspondiente sobre la cartografía).
- Acceso a información ampliada sobre avisos o servicios.
- Ubicación de recursos estáticos.

b) Interconexión con el sistema TrackView (localización de móviles)

Con las funciones siguientes:

- Lista georreferenciada de móviles
- Acceso a información ampliada sobre un móvil

c) Gestión de capas de información

Se han desarrollado servicios web para el soporte de todas las capas que se dibujan sobre la cartografía. Realmente, para cada capa se ha creado un servicio que devuelva un fichero xml o kml con el contenido de la capa. En caso de capas estáticas el servicio puede ser mínimo o nulo accediendo directamente al fichero de salida.

En este momento se disponen de servicios para gestionar:

- Hidrantes
- Parques y retenes
- Cámaras de visualización del tráfico
- Objetos y edificios singulares
- Información meteorológica.
- Avisos

- Servicios

d) Búsquedas georreferenciadas

Para buscar direcciones, carreteras, objetos y topónimos.

e) Gestión de callejeros y bases de datos similares

CAPÍTULO 3. SOFTWARE DE OBTENCIÓN Y TRATAMIENTO DE DATOS

3.1. La nomenclatura EPSG

El antiguo consorcio europeo del petróleo (EPSG), actualmente absorbido por la OGP, realiza desde 1986 varios estudios sobre geodesia por su importancia en la ubicación y medida de los yacimientos petrolíferos. Fruto de estos trabajos es una abundante bibliografía disponible sobre geodesia siendo destacable un dataset público, que van actualizando periódicamente, el cual contiene una colección, especialmente pensada para desarrolladores informáticos, de parámetros de datums, proyecciones, sistemas de referencia de coordenadas (CRS), transformaciones entre estos, etc... los cuales ya se han convertido en un estándar; por esta razón se utiliza la nomenclatura que desarrollaron (nomenclatura EPSG).

3.2. Cambio Coordenadas

Un problema básico de los sistemas informáticos de tratamiento de la cartografía es la conversión de coordenadas entre los diversos sistemas utilizados, teniéndose que implementar algoritmos de transformación entre los diversos tipos de coordenadas (geocéntricas, geográficas y proyectadas) y de transformación entre los diferentes elipsoides utilizados.

En el visor ipSigeMap, al estar pensado para su utilización en Europa, se ha implementado la gestión de los siguientes datum: ED 50, ETRS 89, WGS 84 y NTF (Clarke 1880 IGN Francia). En concreto, el visor permite utilizar los siguientes sistemas de coordenadas, habiéndose implementado todos los algoritmos necesarios para transformarse entre ellos:

- EPSG:4326 (coordenadas geográficas en el datum WGS84)
- EPGS:4258 (coordenadas geográficas en el datum ETRS89)
- EPGS:4230 (coordenadas geográficas en el datum ED50)
- EPGS:4275 (coordenadas geográficas en el datum NTF)
- EPGS:25829 (UTM huso 29 daturm ETRS89)
- EPGS:25830 (UTM huso 30 daturm ETRS89)
- EPGS:25831 (UTM huso 31 daturm ETRS89)
- EPGS:23029 (UTM huso 29 daturm ED50)
- EPGS:23030 (UTM huso 30 daturm ED50)
- EPGS:23031 (UTM huso 31 daturm ED50)
- EPGS:27571 (Lambert zona I datum NTF)
- EPGS:27572 (Lambert zona II datum NTF)
- EPGS:27573 (Lambert zona III datum NTF)
- EPGS:27574 (Lambert zona IV datum NTF)

La realización de cambios de coordenadas sigue el camino del siguiente gráfico.

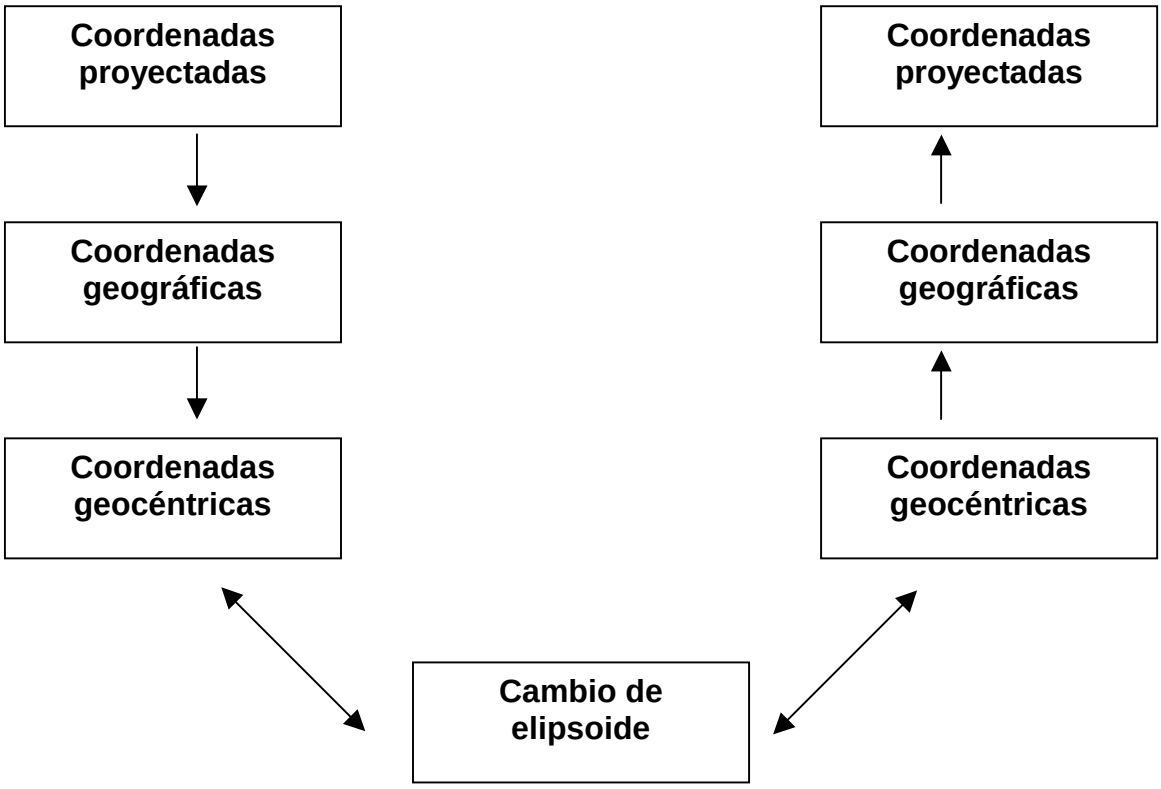

Fig. 3.1 Gráfico cambio de coordenadas

En los anexos se incluyen los algoritmos utilizados

3.3. Altura del terreno

Hasta ahora se ha hablado exclusivamente de dos dimensiones (tanto en coordenadas geográficas como en coordenadas proyectadas), pero en el contexto de este proyecto es necesario contemplar la tercera dimensión, la altura.

Un problema que se presenta al hablar de altura es respecto a que referencia se toma, pudiendo hablarse de altura elipsoidal o de altura sobre el geoide, denominada esta última altura ortométrica que es la utilizada habitualmente (altura sobre el nivel del mar).

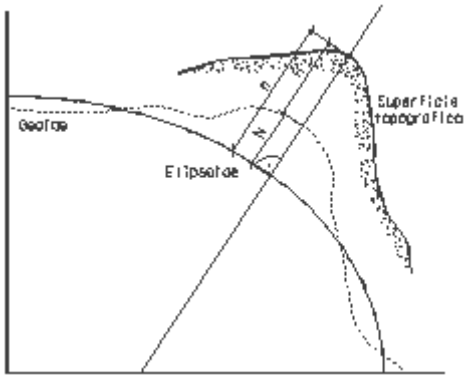

Fig. 3.2 Diferencia de altura entre el geoide y el elipsoide

Para conocer la altura ortométrica del terreno en cada posición se utilizan modelos digitales de elevación (MDE). Estos modelos están disponibles, de forma gratuita, en varias instituciones publicas como el Institut Cartografic de Catalunya, también otras empresas ofrecen sus modelos más precisos pero no gratuitos.

La mayoría de los sistemas se basa en el establecimiento de una malla, dando el valor de la altura en cada casilla de la malla. A efectos prácticos, se suele proporcionar una tabla precedida de una cabecera en la que se especifican la dimensión de cada celda de la malla, el punto origen y el número de filas y columnas de la malla; a continuación se suministra una lista de valores, que son los valores de cada celda escritos por filas.

Para determinar la altura de un punto en concreto hay que determinar primero en que casilla cae el punto y luego en que posición de la tabla está la información de la casilla.

Para la gestión de los MDE se ha desarrollado una clase denominada GeoMdeGlobal, cuyo código completo se incluye en los anexos. Como muestra, en el siguiente listado se muestra la función utilizada para determinar la altura correspondiente a un punto.

```
GeoMdeGlobal.prototype.getAltura = function(punto, scoor)
{
     x = punto.x;
     y = punto.y;

     if ( this.src != scoor)
     {
          cor = transforma(scoor, this.src, [punto.x ,
punto.y]);
          x = parseFloat(cor[0]);
          y = parseFloat(cor[1]);
     }

     c0 = Math.floor((x-this.origenX)/this.paso);
     f0 = Math.floor((this.origenY-y)/this.paso);
```

```
        dx = x - c0*this.paso - this.origenX;
        dy = this.origenY - f0*this.paso - y;

        if ( c0 < 0 || c0 > this.columnas-2 )
            return null;

        if ( f0 < 0 || f0 > this.filas-2 )
            return null;

        pos = f0 * this.columnas + c0;
        h00 = parseFloat(this.grid[pos]);
        h01 = parseFloat(this.grid[pos+1]);
        h10 = parseFloat(this.grid[pos+this.columnas]);
        h11 = parseFloat(this.grid[pos+this.columnas+1]);

        hf0 = h00 + dx*(h01-h00)/this.paso;
        hf1 = h10 + dx*(h11-h10)/this.paso;

        return (hf0 + dy * (hf1-hf0)/this.paso);
};
```

3.4. Datos de navegación

Como ya se explicó anteriormente, los datos de navegación se obtienen a través del GPS, utilizando el protocolo NMEA, en concreto las tres sentencias siguientes:

3.4.1. sentencia $GPGSA

Esta sentencia da información sobre la naturaleza de la posición y el estado de los satélites, como se observa en el siguiente ejemplo:

$GPGSA,A,3,04,05,,09,12,,,24,,,,,2.5,1.3,2.1*39

Donde:

GSA	Satellite status
A	Selección automática de 2D o 3D (M = manual)
3	Posición 3D – los valores incluyen: 1 = sin posición 2 = posición 2D 3 = posición 3D
04,05...	PRNs de los satélites utilizados(12 posibles)
2.5	PDOP incertidumbre 3D
1.3	HDOP incertidumbre 2D
2.1	VDOP incertidumbre en altura
*39	Número de datos enviados

De esta sentencia solo interesa la tercera línea, la cual indica si se está trabajando en 2D o 3D (nivel de precisión en función del número de satélites seguidos).

3.4.2. sentencia $GPGGA

Sentencia esencial para conocer la posición 3D y la exactitud de los datos, a continuación se muestra un ejemplo:

$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47

Donde:

GGA	Global Positioning System Fix Data
123519	Hora a la que fue calculada la posición 12:35:19 UTC
4807.038,N	Latitud 48º 07.038' N
01131.000,E	Longitud 11º 31.000' E
1	Calidad de la posición: 0 = invalida 1 = GPS (SPS) 2 = DGPS 3 = PPS 4 = Real Time Kinematic 5 = Float RTK 6 = estimated 7 = Manual input mode 8 = Simulation mode
08	Número de satélites
0.9	Incertidumbre horizontal
545.4,M	Altitud, metros sobre el nivel del mar
46.9,M	Altura del geoide (al nivel del mar) sobre el elipsoide WGS84
(campo vació)	tiempo en segundos de la ultima actualización DGPS
(campo vació)	ID de la estación DGPS
*47	Número de datos enviados

Con esta sentencia se obtienen la latitud, la longitud, la calidad de la posición, la altura ortométrica y la altura del geoide utilizado.

3.4.3. sentencia $GPRMC

Esta sentencia es la propia versión de NMEA del GPS PVT (posición, velocidad y tiempo), se observa mas abajo un ejemplo:

$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A

Donde:

RMC	Recommended Minimum sentence C
123519	Hora a la que fue calculada la posición 12:35:19 UTC
A	Estado A=activado or V=Void (nulo).
4807.038,N	Latitud 48 deg 07.038' N
01131.000,E	Longitud 11 deg 31.000' E
022.4	Velocidad sobre el suelo, en knots
084.4	Rumbo, en grados
230394	Fecha, 23rd of March 1994
003.1,W	Desviación magnética

*6A Número de datos enviados

De esta sentencia se utiliza la hora, el estado, la latitud, la longitud, la velocidad y el rumbo.

3.5. Tratamiento de datos NMEA

Como ya se ha explicado antes, los datos de longitud y latitud se extraen directamente de la sentencia sin tener que añadir ningún cálculo más. Lo mismo se puede decir respecto a la altura ortométrica.

De la velocidad interesan tres variables, la velocidad sobre el suelo, el rumbo y la velocidad vertical. Tanto la velocidad sobre el suelo como el rumbo se obtienen directamente de la sentencia, pero el cálculo de la velocidad vertical es algo más complejo no proporcionándose directamente, al ser un protocolo de origen marino. Se calcula a partir de dos vectores: el primero guarda las cinco últimas alturas registradas y el segundo en que momento de tiempo se han hecho.

```
public double[] getVelocidad()
     {
     .........
               /// Fecha
          i=5;
          while (i>0)
          {
          gpsdate [i] = gpsdate [i-1];
          i--;
          }
          gpsdate [0] = new Date();

               ///Ahora altura
          i=5;
          while (i>0)
          {
          gpsaltura [i] = gpsaltura [i-1];
          i--;
          }
          gpsaltura [0] = gpshorto;
          i=0;
               ///Ahora calculamos velocidad
          gpsVh=0;
          while (i>4)
          {
          gpsVh +=(gpsaltura[i+1]-
          gpsaltura[i])/(gpsdate[i+1]-gpsdate[i]);
          i++;
          }
               ///calculamos velocidad y lo pasamos a m/s

          gpsVh = (gpsVh*1000)/4;

     .......
     }
```

CAPÍTULO 4. SOFTWARE DE ANÁLISIS DE LAS COLISIONES GEOMÉTRICAS

Para detectar posibles colisiones se calculan posibles intersecciones del rumbo del helicóptero con los obstáculos. Tanto el helicóptero como los posibles obstáculos son tratados como tres tipos de figuras geométricas posibles: Polilínea, Paralepipedo y cilindro.

Se ha desarrollado una librería de clases en javascript para el análisis de las colisiones. Con el objetivo de poder utilizar esta libreria para cualquier otro tipo de situación que se pueda analizar como una colisión de objetos geométricos, esta se ha realizado de la manera más genérica posible.

El helicóptero es tratado como un paralepipido cambiante en función de sus coordenadas, características operativas y velocidad. Esta representación no es representa la geometría real del helicóptero, sino que engloba el área de seguridad necesaria para predecir las colisiones. La única imposición que se añade es que nunca el objeto que representa al helicóptero podrá estar por debajo de los obstáculos , ya que se considera que en ese caso siempre hay peligro de colisión y simplifica mucho los cálculos al poder tratarse, a partir de este primer calculo, todas las colisiones en 2D.

Para representar los distintos objetos geométricos de define la interfaz abstracta GeoObjetoGeometrico que será implementada por las clases Geo3DPolilinea, Geo3DCilindro y Geo3DParalepipedo.

Esta interfaz desciende a su vez de la interfaz GeoSuperposición (que es la que permite dibujar los obstáculos sobre la cartografía).

La lista de obstáculos definidos se representa mediante la clase GeoColeccionObjGeometricos que hereda de la clase GeoColeccion (que representa cualquier colección de objetos visualizables sobre la cartografía, esto es, que implementen la interfaz GeoSuperposicion).

Para determinar la posible colisión del helicóptero con los obstáculos, se analiza la colisión del objeto que representa al helicóptero con cada uno de los objetos de la colección de obstáculos.

Se analizan a continuación los diferentes algoritmos utilizados para la determinación de las posibles colisiones.

4.1. Intersección de paralepípedos envolventes

Con la intención de evitar un máximo de cálculos complejos, en un principio se sustituyen los objetos por sus paralepípedos envolventes y se analiza la posible colisión de estos. En caso de que esta se produzca se pasa al análisis del caso real, determinando si es una falsa alarma o es real.

Ya que para cada objeto, la forma de calcular el paralepípedo envolvente es diferente y dentro del programa pertenece a la clase correspondiente a cada tipo de objeto, esta es explicada mas adelante, en sus correspondientes subapartados.

Los paralepípedos vienen representado por un vector lim [Xmin, Xmax, Ymin, Ymax]. Para calcular la intersección de paralepípedos simplemente hay que ver que la X mínima de uno de los paralepípedos es mas grande que la del otro, pero a su vez mas pequeña que la X máxima, y lo mismo para las Y.

```
GeoObjetoGeometrico.interseccionLimites = function(lim0,
lim1)
{
     return ( lim0[0] < lim1[1] && lim0[1] > lim1[0] &&
lim0[2] < lim1[3] && lim0[3] > lim1[2] );
};
```

En caso de que esta función retorne True, se considera que hay intersección entre los paralepípedos y se pasa a llamar a la intersección del paralepipedo que representa al helicóptero con la polilínea, el cilindro o el paralepípedo que represente a cada obstáculo, en función de cada caso, con las distintas implementaciones de la función abstracta

```
GeoObjetoGeometrico.interseccionLimites(this.limites,
paralep.limites)
```

4.2. Análisis de la colisión paralepípedo-polilínea

Las polilíneas representarán obstáculos que se puedan representar como la unión de puntos con una línea recta, es decir, segmentos rectos consecutivos no alineados (líneas de tendido eléctrico,....). Por lo tanto, para facilitar el cálculo, se tratará cada segmento de forma independiente, excepto para crear el paralepípedo envolvente, que se utilizará la totalidad de la polilínea.

Para evitar paralepípedos envolventes de gran magnitud, lo que provocaría constantes cálculos en el programa, se ha impuesto la prohibición de no representar polilíneas mayores de 10puntos. Las que excedan de estos limites habrá que introducirlas como 2 polilíneas independientes en la base de datos, es decir una polilínea de 15 puntos será introducida como una de 10 puntos y otra de 5.

Cada objeto asociado a la clase Geo3DPolilinea posee una ID o nombre, unos vértices con sus coordenadas y alturas y unas propiedades de color, ancho y opacidad para su dibujo. Además, a través de la función getAlturas también se obtendrá una altura máxima y una altura mínima del conjunto de vértices formado por cada polilínea y como se mostrará a continuación unos limites encontrados con la utilización de la función getLimites.

```
function Geo3DPoliLinea(id, vertices, color, ancho,
opacidad)
{
     this.vertices = vertices;
```

```
     this.color = color;
     this.ancho = ancho;
     this.opacidad = opacidad;

     this.id = id;
     this.limites = this.getLimites();

     var alt = this.getAlturas();
     this.zmin = alt[0];
     this.zmax = alt[1];

     this.svg = null;
     this.objsvg = null;
     this.mapa = null;
}

Geo3DPoliLinea.prototype.getAlturas = function()
{
     var alt = [this.vertices[0].z, this.vertices[0].z];
     for (var iv = 0; iv < this.vertices.length; iv++)
     {
          if ( this.vertices[iv].z < alt[0] )
               alt[0] = this.vertices[iv].z; //zmin

          if ( this.vertices[iv].z > alt[1] )
               alt[1] = this.vertices[iv].z; //zmax
     }

     return alt;
};
```

4.2.1. Getlimites

El algoritmo utilizado para crear el paralepípedo envolvente se basa en ir comparando, uno a uno, todos los puntos de la polilínea con los valores máximos y mínimos de esta encontrados hasta ese momento.

```
Geo3DPoliLinea.prototype.getLimites = function()
{
     var limites = [this.vertices[0].x, this.vertices[0].x,
this.vertices[0].y, this.vertices[0].y];

     for(var i=0 ; i<this.vertices.length ; i++)
     {
          if( this.vertices[i].x < limites[0] )
               limites[0] = this.vertices[i].x;

          if( this.vertices[i].x > limites[1] )
               limites[1] = this.vertices[i].x;

          if( this.vertices[i].y < limites[2] )
               limites[2] = this.vertices[i].y;

          if( this.vertices[i].y > limites[3] )
               limites[3] = this.vertices[i].y;
     }
```

```
     return limites;          //cubo=Xmin,Xmax,Ymin,Ymax
vector
};
```

4.2.2. Determinación de la posible colisión

Una vez se ha detectado intersección de los paralepípedos envolventes, se analiza la polilínea segmento a segmento.

```
Geo3DPoliLinea.prototype.interseccionParalepipedoGen =
function(paralep)
{
     var ret = false;

     if( this.zmax >= paralep.zmin &&
GeoObjetoGeometrico.interseccionLimites(this.limites,
paralep.limites) )
     {
          for (var is=0 ; is<this.vertices.length-1 ; is++)
               if (
paralep.interseccionSegmentoGen(this.vertices[is],
this.vertices[is+1] ) )
               {
                    ret = true;
                    break;
               }
     }

     return ret;
};
```

Lo primero que hace es determinar un paralepípedo mas pequeño, con cada segmento, y volver a comprobar si hay intersección, en caso positivo se calcula la ecuación de la recta de este segmento y el punto de intersección de esta con las 4 aristas de la proyección del paralepipedo, comprobando si estos puntos están dentro del segmento y del perímetro del paralepipedo y en caso afirmativo genera una alarma de colisión.

$$xc=(k2-k1)/(m1-m2) \quad \textbf{(4.1)}$$

$$yc=(k2*m1-k1*m2)/(m1-m2) \quad \textbf{(4.2)}$$

Si alguna de las rectas es vertical el sistema fallaría produciendo una división por 0, para evitar esto se comprueba que ninguna de ellas tenga la misma x inicial y final y en caso afirmativo se le asigna a xc (punto x del cruce) este valor repetido.

4.3. Análisis de la colisión paralepipedo- paralepipedo

Los paralepípedos representan obstáculos con base rectangular y cierta altura, normalmente edificios. Para simplificar los cálculos todos han de ser paralepidedos ortogonales, es decir, todas sus bases han de ser rectángulos.

Cada objeto asociado a la clase Geo3DParalepipedo posee una ID o nombre, cuatro vértices con sus coordenadas, las alturas máxima y mínima de cada objeto y unas propiedades de color y opacidad. Además internamente se mantendrán variables para almacenar el centro y los límites obtenidos con la función getLimites.

```
function Geo3DParalepipedo(id, vertices, zmin, zmax, color,
opacidad)
{
     this.vertices = vertices;
     this.zmin = zmin;
     this.zmax = zmax;
     this.color = color;
     this.opacidad = opacidad;

     this.id = id;
     this.limites = this.getLimites();

     this.centro = [(this.limites[0] + this.limites[1])/2,
(this.limites[2] + this.limites[3])/2];

     this.svg = null;
     this.objsvg = null;
     this.mapa = null;
}
```

4.3.1. Getlimites

El cubo, en este caso, se calcula exactamente igual que en el caso de la polilínea explicado anteriormente.

4.3.2. Determinación de la posible colisión

Para el calculo se utiliza la misma función que para el cálculo de la intersección con la polilínea, analizando esta vez las cuatro aristas del rectángulo base del paralepípedo.

```
Geo3DParalepipedo.prototype.interseccionParalepipedoGen =
function(paralep)
{
     var ret = false;

     if( this.zmax >= paralep.zmin &&
GeoObjetoGeometrico.interseccionLimites(this.limites,
paralep.limites) )
     {
          for (var is=0 ; is<4; is++)
```

```
          {
               p1 = this.vertices[is];
               if ( is < 3 )
                    p2 = this.vertices[is+1];
               else
                    p2 = this.vertices[0];

               if ( paralep.interseccionSegmentoGen(p1, p2)
)
               {
                    ret = true;
                    break;
               }
          }
     }
```

4.4. Análisis de la colisión paralepípedo-cilindro

Los cilindros representan obstáculos en un punto fijo, como son antenas, torres...,

Cada objeto asociado a la clase Geo3DCilindro posee una ID o nombre, dos vértices con sus coordenadas, las alturas máxima y mínima de cada objeto y unos parámetros de color y opacidad. Además internamente se mantendrán variables para almacenar el centro y los límites obtenidos con la función getLimites.

Para tener en cuenta el posible cambio del sistema de referencia, que podría implicar que la base del cilindro pasase a ser una elipse se almacenan los dos posibles radios.

```
function Geo3DCilindro(id, vertices, zmin, zmax, color,
opacidad)
{
     this.vertices = vertices;
     this.zmin = zmin;
     this.zmax = zmax;
     this.color = color;
     this.opacidad = opacidad;

     this.id = id;

     this.svg = null;
     this.objsvg = null;
     this.mapa = null;

     this.centro = new GeoPunto(this.vertices[0].x,
this.vertices[1].y);
     this.radiox = this.vertices[0].y - this.vertices[1].y;
     this.radioy = this.vertices[1].x - this.vertices[0].x;
     this.limites = this.getLimites();
}
```

4.4.1. Getlimites

El algoritmo utilizado para crear el cubo se basa en sumar y restar el radio al centro de éste.

```
Geo3DCilindro.prototype.getLimites = function()
{
     return [this.centro.x - this.radioy, this.centro.x +
this.radioy, this.centro.y - this.radiox, this.centro.y +
this.radiox];};
```

4.4.2. Calculo intersección subapartado

Para calcular la intersección entre las proyecciones de un paralepípedo y un cilindro se calcula la ecuación de la recta resultante en la unión de ambos centros y la distancia.

Para cada arista de la base del paralepípedo se calcula el punto de cruce con la recta de unión de los centros y la distancia del cruce al centro del cilindro.

$$d(A,B) = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2} \qquad \textbf{(4.3)}$$

Llegados a este punto, una forma sencilla de calcular si hay intersección es comprobar si la distancia de cruce es menor que el radio, de ser así hay intersección, pero esto genera error en paralepípedos muy alargados como se observa en la fig. 4.1, no indicando intersección cuando si la hay.

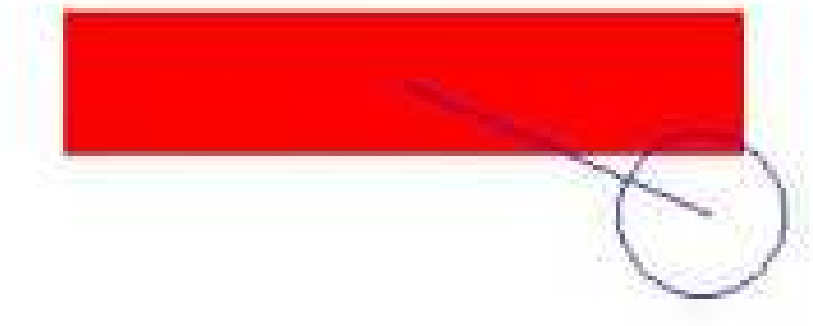

Fig. 4.1 Error común en paralepípedos alargados

Para solventar este problema se calcula la distancia del centro del cilindro al segmento y en caso de ser menor que el radio hay intersección.

Para calcular la distancia de un punto a un segmento lo primero es averiguar la distancia a la recta del segmento (fórmula 4.5) y luego averiguar si el lugar de cruce esta dentro o fuera del segmento.

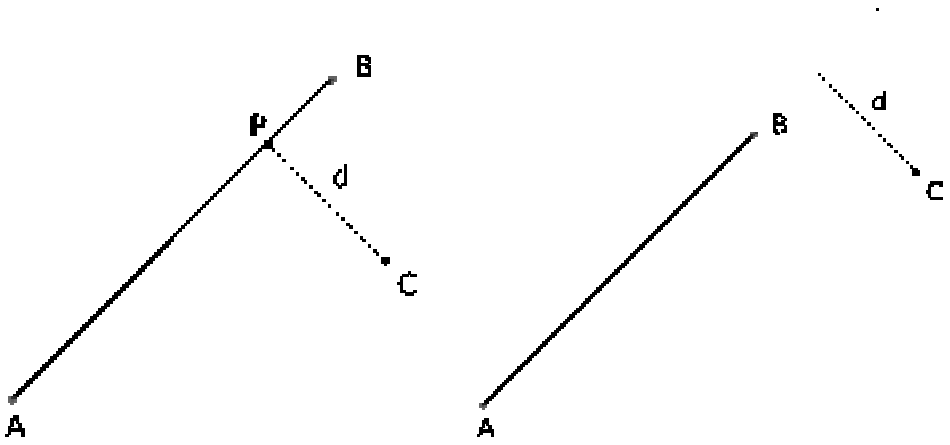

Fig. 4.2 Distancia a la recta de un segmento desde un punto

En caso de estar fuera del segmento, la distancia del punto al segmento será la distancia del punto al punto A o al punto B del segmento en función de si éste ha sobrepasado o no ha llegado al segmento. Para ello se utilizará la fórmula 4.4.

$$u = \frac{(C_x - A_x)(B_x - A_x) + (C_y - A_y)(B_y - A_y)}{(B_x - A_x)^2 + (B_y - A_y)^2} \quad \textbf{(4.4)}$$

Si el valor de ***u*** esta entre 0 y 1 el punto está dentro del segmento, si es mayor de 1 el punto ha sobrepasado al segmento por B y si es menor de 0 el punto no ha llegado al segmento.

$$d = \frac{(B_x - A_x)(C_y - A_y) - (B_y - A_y)(C_x - A_x)}{\sqrt{(B_x - A_x)^2 + (B_y - A_y)^2}} \quad \textbf{(4.5)}$$

Por último esta solución es errónea en el caso de segmentos verticales, en cuyo caso, al igual que se explicaba anteriormente, el valor xc pasa a ser el de la vertical.

```
Geo3DCilindro.prototype.interseccionParalepipedoGen =
function(paralep)
{
```

```
     var ret = false;
     var exint = false;

     if( this.zmax >= paralep.zmin &&
GeoObjetoGeometrico.interseccionLimites(this.limites,
paralep.limites) && this.centro )
     {
          var distanciacentros =
Math.sqrt((paralep.centro[0]-
this.centro.x)*(paralep.centro[0]-
this.centro.x)+(paralep.centro[1]-
this.centro.y)*(paralep.centro[1]-this.centro.y));

          this.radio = this.radiox;
          if ( this.radioy > this.radiox )
               this.radio = this.radioy;

          //calculo pendiente recta que une los centros
          var m1, m2, k1, k2;
          var x1, y1, x2, y2, xc, yc;
          var v1 = false;
          var v2 = false;
          var ladosegmento, distpuntseg;

          if( paralep.centro[0] != this.centro.x )
          {
               m1 = (paralep.centro[1]-this.centro.y) /
(paralep.centro[0]-this.centro.x);
               k1 = this.centro.y - m1 * this.centro.x;
          }
          else
               v1 = true;

          for(var i=0;i<4;i++)
          {
               v2 = false;
               x1 = paralep.vertices[i].x;
               y1 = paralep.vertices[i].y;

               if ( i == 3 )      //del vertice 3 al 0,
cerrar rectangulo
               {
                    x2 = paralep.vertices[0].x;
                    y2 = paralep.vertices[0].y;
               }
               else //resto de rectas
               {
                    x2 = paralep.vertices[i+1].x;
                    y2 = paralep.vertices[i+1].y;
               }

               if( x1 != x2 )
               {
                    m2 = (y1 - y2) / (x1 - x2);
                    k2 = y2 - m2 * x2;
               }
               else
                    v2 = true;
```

```
                if ( v1 && v2 )
                     continue;

                if ( !v1 && !v2 && m1==m2 )
                     continue;

                //busco punto cruce
                if ( v1 )
                {
                     xc = this.centro.x;
                     yc = m2 * xc + k2;
                }
                else
                     if ( v2 )
                     {
                          xc = x1;
                          yc = m1 * xc + k1;
                     }
                     else
                     {
                          xc=(k2-k1)/(m1-m2);
                          yc=(k2*m1-k1*m2)/(m1-m2);
                     }

                //compruebo que el punto de cruce esta
dentro de la recta (solo hace falta comprobarlo para la x o
la y)
                if ( v1 )
                {
                     if ( paralep.centro[1] < this.centro.y
)
                     {
                          if ( yc < paralep.centro[1] ||
this.centro.y < yc )
                               continue;
                     }
                     else
                          if ( yc > paralep.centro[1] ||
this.centro.y > yc )
                               continue;

                }
                else
                {
                     if ( paralep.centro[0] < this.centro.x
)
                     {
                          if ( xc < paralep.centro[0] ||
this.centro.x < xc )
                               continue;
                     }
                     else
                          if ( xc > paralep.centro[0] ||
this.centro.x > xc )
                               continue;
                }
```

```
                if ( v2 )
                {
                     if ( y1 < y2 )
                     {
                          if ( yc < y1 || y2 < yc )
                               continue;
                     }
                     else
                          if ( yc > y1 || y2 > yc )
                               continue;
                }
                else
                {
                     if ( x1 < x2 )
                     {
                          if ( xc < x1 || x2 < xc )
                               continue;
                     }
                     else
                          if ( xc > x1 || x2 > xc )
                               continue;
                }

                exint = true;
                distanciacruce = Math.sqrt((this.centro.x-
xc)*(this.centro.x-xc)+(this.centro.y-yc)*(this.centro.y-
yc));

                if ( distanciacruce > distanciacentros )
                     continue;

                if ( v2 )
                {
                     ladosegmento = 0.5;

                     if ( y1 < y2 )
                     {
                          if ( this.centro.y < y1 )
                               ladosegmento = -1;

                          if ( this.centro.y > y2 )
                               ladosegmento = 2;
                     }
                     else
                     {
                          if ( this.centro.y > y1 )
                               ladosegmento = -1;

                          if ( this.centro.y < y2 )
                               ladosegmento = 2;
                     }

                }
                else
                     ladosegmento = ((this.centro.x-x1)*(x2-
x1)+(this.centro.y-y1)*(y2-y1))/((x2-x1)*(x2-x1))+((y2-
y1)*((y2-y1)));
```

```
                if( ladosegmento <= 1 )
                {
                     if( ladosegmento >= 0 )
                     {
                          distpuntseg =((x2-
x1)*(this.centro.y-y1)-(y2-y1)*(this.centro.x-
x1))/(Math.sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1)));
                     }
                     else
                     {
                          distpuntseg =
Math.sqrt((this.centro.x-x1)*(this.centro.x-
x1)+(this.centro.y-y1)*(this.centro.y-y1));
                     }
                }
                else
                {
                     distpuntseg = Math.sqrt((this.centro.x-
x2)*(this.centro.x-x2)+(this.centro.y-y2)*(this.centro.y-
y2));
                }

                if ( distpuntseg < this.radio )
                     ret=true;

                break;
           }

           if ( !exint )
                ret = true;

     }

     return ret;
};
```

CAPÍTULO 5. SOFTWARE DE AMPLIACIÓN DE LA INTERFAZ DE USUARIO DEL VISOR

A la interfaz visual del programa se le han añadido varias opciones como son el control del helicóptero, control de posición, la visualización de obstáculos o la ampliación de la visualización de las coordenadas.

5.1. Visualización de las coordenadas

El visor dispone de un control para la visualización de las coordenadas del punto sobre el que está el ratón en la parte superior de la pantella. A esta herramienta se le ha añadido la posibilidad de que indique también la altura del terreno, para ello lee los datos a partir del mde del punto donde se encuentre, a través de la función GetAltura explicada anteriormente,.

En función del tipo de cartografía utilizada nos mostrará la posición en una u otra coordenada, siempre indicándonos el tipo de proyección que se está utilizando.

Como se puede observar en la figura 5.1, nos indica que para la zona del parking de la Escola Politècnica Superior de Castelldefels, (coordenadas 41º16’34,74”N 1º59’17,40”O en el sistema WGS84) la altura es de 2,18 metros.

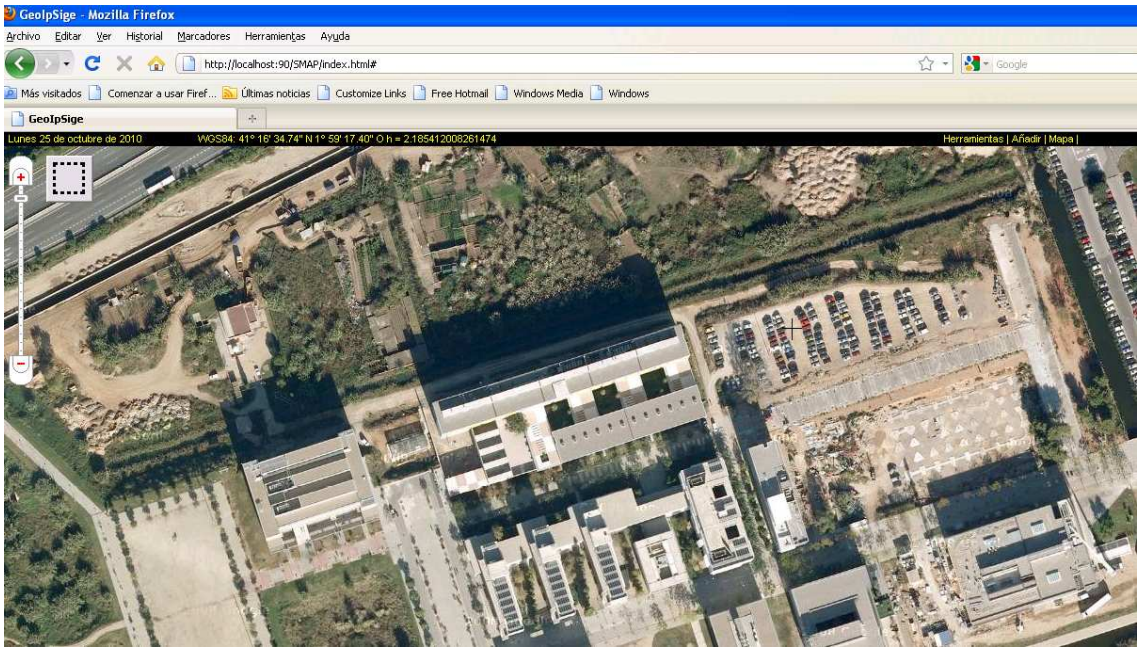

Fig. 5.1 Vista de la Escola Politècnica Superior de Castelldefels desde el visor IpSigeMap

5.2. Control del helicóptero

El objeto fundamental de este proyecto se muestra en un control que se ha desarrollado ("control del helicóptero")

Este control recibe eventos de posición generados por el corazón del sistema al leer el GPS. Al recibir un evento, se redibuja el helicóptero y se controlan las posibles colisiones.

Como se puede observar en la figura 5.2, la interfaz es muy amena y posee un botón, similar al de los reproductores musicales, que permite tener el sistema de alarmas conectado o desconectado. El sistema proporciona información sobre la posición, la altura y la velocidad sobre el suelo del helicóptero.

En caso de peligro de colisión las alarmas pertinentes se pondrán en rojo, ya sea la alarma de proximidad al suelo o la de proximidad a obstáculos.

La posición del helicóptero es mostrada en el tipo de coordenadas que corresponda en función del tipo de cartografía utilizada.

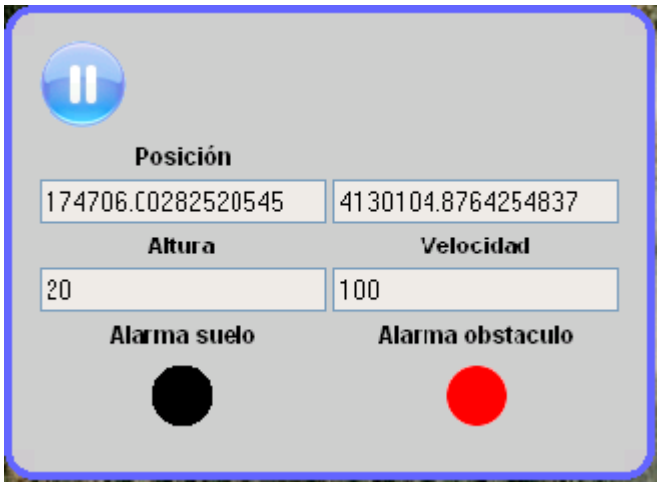

Fig. 5.2 “control del helicóptero” en el visor IpSigeMap

5.3. Visualización de obstáculos

Los obstáculos se definen en un documento de texto en formato JSON (para su lectura automática desde javascript). Lo primero que se introduce en ellos es el sistema de referencia utilizado, seguido de la definición de los obstáculos. Para cada obstáculo se ha de indicar de que tipo es (polilínea, paralepípedo o cilindro), una id o nombre y ciertos parámetros en función del tipo de obstáculo como son sus vértices, alturas máximas de cada zona,...

A continuación se muestra un ejemplo:

{srf: 'EPSG:23030',
objetos:[{tipo:1,id:"obj1",vert:[[162506,4130235,12], [172506,4140235,15]] [172506,4140235,15], [172526,4140275,25]] },
{tipo:2,id:"obj2",vert:[[175206,4130355],[175326,4130235]],zmin: 12,zmax: 15},
{tipo:3,id:"obj3",vert:[[175506,4130235], [175706,4130235], [175706,4130435], [175506,4130435]],zmin:12,zmax:15}]
}

Tipo 1 se refiere a una polilínea, de la cual se indica los puntos de cada vértice y su altura, el tipo 2 indica que se trata de un cilindro, y se añaden los dos puntos que lo definen y la altura máxima y mínima del cilindro y el tipo 3 indica que es un paralepípedo y muestra los cuatro vértices de este y su altura máxima y mínima.

Estos obstáculos son dibujados en el visor cartográfico en color azul y siempre que sea necesario, en función del tipo de cartografía utilizada, se realizará un cambio de coordenadas.

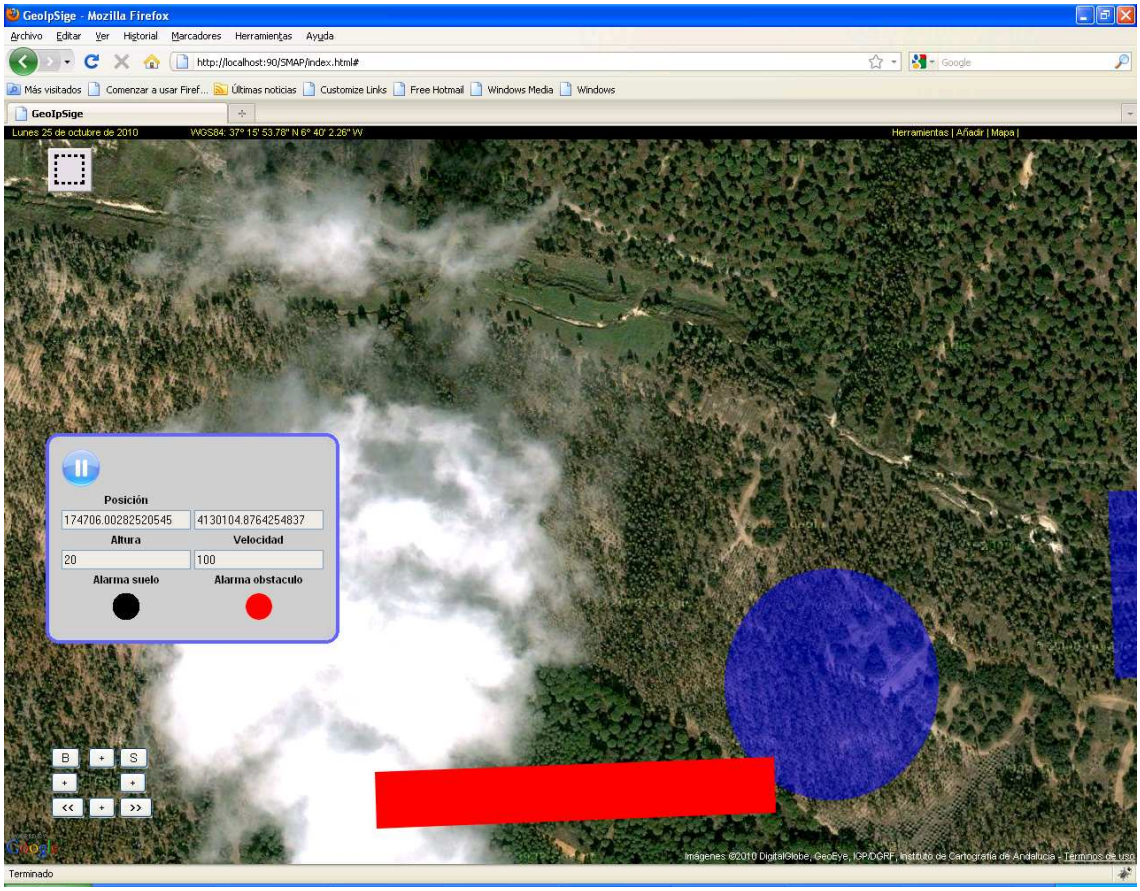

Fig. 5.3 Alarma de posible colisión

5.4. Control de posición

Para testeo de las nuevas funciones implementadas, se ha añadido el control de posición, que es la herramienta encargada de simular los datos suministrados por el GPS. Se puede variar la posición del helicóptero, la altura (botones B,S) y la velocidad (botones << y >>) y así poder comprobar el correcto funcionamiento sin necesidad de ningún helicóptero real.

Fig. 5.4 Control de posición en el visor IpSigeMap

CAPÍTULO 6. CONCLUSIONES

Se han obtenido los resultados que se pretendían, desarrollo de un sistema de ayuda a la navegación para helicópteros, y que fueron expuestos en la presentación del proyecto. En concreto:

1. Se ha aprendido a programar en los lenguajes Javascript y HTML a través de los cuales se han podido implementar las nuevas funciones del visor.
2. Se ha añadido la dimensión Z al visor cartográfico IpSigeMap.
3. Se ha definido la interfaz abstracta de objetos geométricos y se ha implementado con las clases Geo3DPolilinea, Geo3DCilindro y Geo3DParalepipedo.
4. Se ha investigado y creado algoritmos de colisión entre polilíneas, paralepípedos y cilindros.
5. Se ha sabido extraer y encontrar información a partir de receptores GPS, tales como la localización de un móvil, su velocidad horizontal, velocidad vertical, altitud o rumbo, todo ello con la utilización del protocolo NMEA, pudiendo obtener las variables de ubicación y desplazamiento en tiempo real del helicóptero.
6. Se han utilizado modelos digitales de elevación del terreno MDE.
7. Se han Implementado utilidades de control y aviso de alarmas en el visor IpSigeMap los cuales permiten ver la posición del helicóptero y de los obstáculos en todo momento y avisa en caso de peligro de colisión con estos o con el suelo.
8. Se ha creado un control de manipulación de la posición que simula un objeto helicóptero para el testeo de las nuevas características del visor cartográfico.

REFERENCIAS

[1] Flanagan, D., *JavaScript La Guía Definitiva*, Ed. ANAYA O'REILLY, 2006.
[2] Salazar Hernández, D.J., *Navegación Aérea, Cartografía y Cosmografía*, 2008.
[3] Sainz de Baranda, J.S., *Sistema ipSigeMap Documentación técnica*, 2010.
[4] Goizueta, J., *Sistemas de referencia Geodésicos,* ECAS Técnicos Asociados S.A.
[5] Alonso Fernández-Coppel, I., *Localizaciones geográficas. Las coordenadas geográficas y la proyección UTM,* Universidad de Valladolid, 2001.
[6] http://www.epsg.org/
[7]http://www.fomento.es/MFOM/LANG_CASTELLANO/DIRECCIONES_GENERALES/INSTITUTO_GEOGRAFICO/
[8] http://www.icc.cat/
[9] http://spatialreference.org/
[10] http://luisrey.wordpress.com/
[11] http://www.gpsinformation.org/dale/nmea.htm
[12] http://www.solocodigo.com/index.php
[13] http://code.google.com/intl/es/apis/maps/documentation/javascript/

ANEXOS

TÍTULO DEL TFC: Desarrollo de un sistema de ayuda a la navegación para helicópteros

TITULACIÓN: Ingeniería Técnica Aeronáutica, especialidad Aeronavegación

AUTOR: Gonzalo Sainz de Baranda Garrido

DIRECTOR: José Manuel Sainz de Baranda Graf

FECHA: 30 de noviembre de 2010

ANEXO 1. FORMULAS DE CAMBIOS DE COORDENAS

```
/* cálculos geográficos */
/*
 * Devuelve la representación de un punto en coordenadas geográficas
en formato legible
 */
function geoToString(lt, ln)
{
                  var lat = lt;
                  var lon = ln;
                  var olat = "N";
                  var olon = "E";
                  if ( lat <0 )
                  {
                        lat = -lat;
                        olat = "S";
                  }
                        if ( lon < 0 )
                        {
                              lon = -lon;
                              olon = "W";
                        }

                  var ltgr = Math.floor(lat);
                  var ltmi = Math.floor( (lat-ltgr)*60 );
                  var ltse = (lat - ltgr) * 3600 - ltmi * 60;
                  ltse = ltse.toFixed(2);

                  var lngr = Math.floor(lon);
                  var lnmi = Math.floor( (lon-lngr)*60 );
                  var lnse = (lon - lngr) * 3600 - lnmi * 60;
                  lnse = lnse.toFixed(2);

                  return ltgr + "º " + ltmi + "' " + ltse + "\" " +
olat + "    " + lngr + "º " + lnmi + "' " + lnse + "\" " + olon;
};



/**
 * determina el huso UTM
 * @param longr: longitud en grados
 * @return
 */
function detHuso(longr)
{
      return Math.floor(longr/6 + 31);
};


/**
 * Convierte de grados a radianes
 * @param grados
 * @return
 */
function grArad(grados)
{
      return (grados * Math.PI/180.0);
};
```

```
/**
 * Convierte de radianes a grados
 * @param rad
 * @return
 */
function radAgr(rad)
{
	return (rad * 180.0 / Math.PI);
};



/**
 * Normaliza al sistema WGS84 coordenadas geograficas en otro
elipsoide
 * @param (number): latitud (radianes)
 * @param (numbar): longitud (radianes)
 * @param (number): elipsoide
 * @return ([numero, numero]) : [longitud, latitud] (grados)
 */
function normalizaGeograficas(lat, lon, el)
{
	var geo = new GeoGeograficas(lat, lon, 0, el);
	var cart = ELP_geograficasAcartesianas(geo);
	var cartnorm = ELP_normaliza(el, cart, 1);
	var geonorm = ELP_cartesianasAgeograficas(cartnorm);

	return [radAgr(geonorm.lon), radAgr(geonorm.lat)];
};

/**
 * Convierte del sistema WGS84 a coordenadas geograficas en otro
elipsoide
 * @param (number): latitud WGS84 (grados)
 * @param (numbar): longitud WGS84 (grados)
 * @param (number): elipsoide destino
 * @return ([numero, numero]) : [longitud, latitud] (radianes)
 */
function desnormalizaGeograficas(lat, lon, el)
{
	var geo = new GeoGeograficas(grArad(lat), grArad(lon), 0, 0);
	var cart = ELP_geograficasAcartesianas(geo);
	var cartdnorm = ELP_normaliza(el, cart, -1);
	var geodnorm = ELP_cartesianasAgeograficas(cartdnorm);

	return [geodnorm.lon, geodnorm.lat];
};



/**
 * Normaliza al sistema WGS84 coordenadas UTM
 * @param (numero): x
 * @param (numero): y
 * @param (numero): huso
 * @param (numero): elipsoide
 * @return ([numero, numero]) : [longitud, latitud]
 */
function normalizaUTM(x, y, huso, el)
{
```

```
	var utm = new GeoCoordPlanas(x, y, 0, huso, el);
	var geo = ELP_utmAgeograficas(utm);
	return normalizaGeograficas(geo.lat, geo.lon, el);
};


/**
 * Convierte del sistema WGS84 a coordenadas UTM en otro sistema
 * @param (numero): latitud WGS84 (grados)
 * @param (numero): longitud WGS84 (grados)
 * @param (numero): huso de la proyección UTM destino
 * @param (numero): elipsoide destino
 * @return ([numero, numero]) : [x, y] (metros)
 */
function desnormalizaUTM(lat, lon, huso, el)
{
	var dgeo = desnormalizaGeograficas(lat, lon, el);
	var geo = new GeoGeograficas(dgeo[1], dgeo[0], 0, el);
	var utm = ELP_geograficasAutm(geo, huso);
	return [utm.x, utm.y];
};


/**
 * Normaliza al sistema WGS84 coordenadas lambert
 * @param (numero): x
 * @param (numero): y
 * @param (numero): zona
 * @param (numero): elipsoide
 * @return ([numero, numero]) : [longitud, latitud]
 */
function normalizaLambert(x, y, zona, el)
{
	var lam = new GeoCoordPlanas(x, y, 0, zona, el);
	var geo = ELP_lambertAgeograficas(lam);
	return normalizaGeograficas(geo.lat, geo.lon, el);
};


/**
 * Convierte del sistema WGS84 a coordenadas Lambert en otro sistema
 * @param (numero): latitud WGS84 (grados)
 * @param (numero): longitud WGS84 (grados)
 * @param (numero): zona de la proyección destino
 * @param (numero): elipsoide destino
 * @return ([numero, numero]) : [x, y] (metros)
 */
function desnormalizaLambert(lat, lon, zona, el)
{
	var dgeo = desnormalizaGeograficas(lat, lon, el);
	var geo = new GeoGeograficas(dgeo[1], dgeo[0], 0, el);
	var lam = ELP_geograficasAlambert(geo, zona);
	return [lam.x, lam.y];
};


/**
 * Funciones de normalización de coordenadas (sistema referencia:
WGS84 geográficas)
 *    las coordenadas se envian en formato [x,y], [lon, lat](en
grados)
```

```
*/


/**
 * Normaliza coordenadas desde el SRS origen a WGS84
 * @param (literal): codigo del sistema de referencia origen
 * @param ([numero, numero]): coordenadas origen
 * @return ([numero, numero]) : [longitud, latitud] (grados)
*/
function normalizaCoordenadas(sref, coord)
{
      var xn = coord[0];
      var yn = coord[1];

      if ( sref == "EPSG:4326" )
      {
      }

      if ( sref == "EPSG:4326INV" )
      {
            xn = coord[1];
            yn = coord[0];
      }

      if ( sref == "EPSG:4258" )
      {
            ret = normalizaGeograficas(grArad(yn), grArad(xn), 3);
            xn = ret[0];
            yn = ret[1];
      }

      if ( sref == "EPSG:4258INV" )
      {
            ret = normalizaGeograficas(grArad(xn), grArad(yn), 3);
            xn = ret[0];
            yn = ret[1];
      }

      if (sref == "EPSG:4230" )
      {
            ret = normalizaGeograficas(grArad(yn), grArad(xn), 1);
            xn = ret[0];
            yn = ret[1];
      }

      if (sref == "EPSG:4275" )
      {
            ret = normalizaGeograficas(grArad(yn), grArad(xn), 2);
            xn = ret[0];
            yn = ret[1];
      }


      if (sref == "EPSG:23030" )
      {
            ret = normalizaUTM(xn, yn, 30, 1);
            xn = ret[0];
            yn = ret[1];
      }

      if (sref == "EPSG:23030INV" )
```

```
	{
		ret = normalizaUTM(yn, xn, 30, 1);
		xn = ret[0];
		yn = ret[1];
	}


	if (sref == "EPSG:23031" )
	{
		ret = normalizaUTM(xn, yn, 31, 1);
		xn = ret[0];
		yn = ret[1];
	}


	if (sref == "EPSG:25830" )
	{
		ret = normalizaUTM(xn, yn, 30, 3);
		xn = ret[0];
		yn = ret[1];
	}


	if (sref == "EPSG:27573" )
	{
		ret = normalizaLambert(xn, yn, "III", 2);
		xn = ret[0];
		yn = ret[1];
	}

	return [xn,yn];
};



/**
 * Desnormaliza coordenadas desde WGS84 a un SRS destino
 * @param (literal): codigo del sistema de referencia destino
 * @param ([numero, numero]): [longitud, latitud] WGS84 (grados)
 * @return ([numero, numero]) : coordenadas destino ([x,y] metros o
[lon,lat] grados)
*/
function desNormalizaCoordenadas(sref, coord)
{
	var xn = coord[0];
	var yn = coord[1];
	var ret = null;

	if ( sref == "EPSG:4326" )
	{
		ret = [xn,yn];
	}

	if ( sref == "EPSG:4326INV" )
	{
		ret = [yn,xn];
	}

	if ( sref == "EPSG:4258" )
	{
		coord = desnormalizaGeograficas(yn, xn, 3);
```

```
            ret = [radAgr(coord[0]), radAgr(coord[1])];
      }

      if (sref == "EPSG:4230" )
      {
            coord = desnormalizaGeograficas(yn, xn, 1);
            ret = [radAgr(coord[0]), radAgr(coord[1])];
      }

      if (sref == "EPSG:4275" )
      {
            coord = desnormalizaGeograficas(yn, xn, 2);
            ret = [radAgr(coord[0]), radAgr(coord[1])];
      }

      if (sref == "EPSG:23030" )
      {
            ret = desnormalizaUTM(yn, xn, 30, 1);
      }

      if (sref == "EPSG:23030INV" )
      {
            coord = desnormalizaUTM(yn, xn, 30, 1);
            ret = [coord[1], coord[0]];
      }

      if (sref == "EPSG:23031" )
      {
            ret = desnormalizaUTM(yn, xn, 31, 1);
      }

      if (sref == "EPSG:25830" )
      {
            ret = desnormalizaUTM(yn, xn, 30, 3);
      }


      if (sref == "EPSG:27573" )
      {
            ret = desnormalizaLambert(yn, xn, "III", 2);
      }

      return ret;
};


function transforma(orig, dest, coord)
{
      if ( orig != dest )
      {
            var etrs89 = normalizaCoordenadas(orig, coord);
            return desNormalizaCoordenadas(dest, etrs89);
      }
      else
            return coord;
};

function isCoordenadas(valor)
{
      var ret = false;
      numeros = valor.split(",");
```

```
	if ( numeros.length == 2 )
	{
		ret = true;
		for (var i = 0 ; i < numeros.length ; i++ )
			if (isNaN(parseFloat(numeros[i])) )
			{
				ret = false;
				break;
			}
	}

	return ret;
};

/**
 * Nuevas rutinas de conversion
 */




/*
 * Datos elipsoides
 * Se almacenan en cuatro vectores
 * ELP_A : semieje mayor
 * ELP_F : inversa del aplastamiento  1/f = a / (a-b)
 *
 * ELP_DX... ELP_DS : parámetros tranformación de Bursa-Wolf para
pasar del elipsoide a WGS84
 * Trasformaciones en metros, angulos en segundos y escala en p.p.m.
 */

var ELP_A = [];
var ELP_F = [];
var ELP_DX = [];
var ELP_DY = [];
var ELP_DZ = [];
var ELP_RX = [];
var ELP_RY = [];
var ELP_RZ = [];
var ELP_DS = [];

/*
 * 0 = WGS84 (EPSG:4326)
 */

ELP_A[0] = 6378137.0 ;
ELP_F[0] = 298.257223563;

ELP_DX[0] = 0;
ELP_DY[0] = 0;
ELP_DZ[0] = 0;
ELP_RX[0] = 0;
ELP_RY[0] = 0;
ELP_RZ[0] = 0;
ELP_DS[0] = 0;

/*
 * 1 = ED50 (EPSG:4230)
 */
```

```
ELP_A[1] = 6378388.0;
ELP_F[1] = 297.0;

ELP_DX[1] = -131.032;
ELP_DY[1] = -100.251;
ELP_DZ[1] = -163.354;
ELP_RX[1] = 1.2438;
ELP_RY[1] = 0.0195;
ELP_RZ[1] = 1.1436;
ELP_DS[1] = 9.39;

/*
 * 2 = NTF (Clarke 1880 IGN Francia) (EPSG:4275)
 */

ELP_A[2] = 6378249.2;
ELP_F[2] = 293.466021293627;

ELP_DX[2] = -168;
ELP_DY[2] = -60;
ELP_DZ[2] = 320;
ELP_RX[2] = 0;
ELP_RY[2] = 0;
ELP_RZ[2] = 0;
ELP_DS[2] = 0;

/*
 * 3 = ETRS89 (GRS 1980) (EPSG:4258)
 */

ELP_A[3] = 6378137.0;
ELP_F[3] = 298.257222101;

ELP_DX[3] = 0;
ELP_DY[3] = 0;
ELP_DZ[3] = 0;
ELP_RX[3] = 0;
ELP_RY[3] = 0;
ELP_RZ[3] = 0;
ELP_DS[3] = 0;




/**
 * Clases auxiliares
 */



/**

************************************************************************
********
 * Clase GeoCartesianas : representa un punto geográfico en coordenaas
cartesianas (en metros)

************************************************************************
********
 */
```

```
/**
 * Constructor
 * @param {número} x
 * @param {número} y
 * @param {número} z
 * @param {número} el: Datum (código interno), por defecto 0 (WGS84)
 */
function GeoCartesianas(x, y, z, el)
{
      this.x = x;
      this.y = y;
      this.z = z;
      this.el = el || 0;
};


/**
 * Devuelve una representación del punto como literal
 */
GeoCartesianas.prototype.toString = function()
{
      return "Objeto GeoCartesianas (" + this.x + " , " + this.y + " ,
" + this.z + " , elipsoide:" + this.el + ")";
};


/**
 * Devuelve una representacion del punto como vector de coordenadas
 */
GeoCartesianas.prototype.toVector = function()
{
      return [this.x, this.y, this.z];
};




/**

************************************************************************
********
 * Clase GeoGeograficas : representa un punto geográfico en coordenaas
geográficas:
 *  latitud (radianes), longitud (radianes) y altura elipsoidal
(metro)

************************************************************************
********
 */


/**
 * Constructor
 * @param {número} lat
 * @param {número} lon
 * @param {número} he: altura elipsoidal, por defecto 0
 * @param {número} el: Datum (código interno), por defecto 0 (WGS84)
 */
function GeoGeograficas(lat, lon, he, el)
{
```

```
	this.lat = lat;
	this.lon = lon;
	this.he = he || 0;
	this.el = el || 0;
};


/**
 * Devuelve una representación del punto como literal
 */
GeoGeograficas.prototype.toString = function()
{
	return "Objeto GeoGeograficas (" + geoToString(radAgr(this.lat),
radAgr(this.lon)) + " , elipsoide:" + this.el + ")";
};


/**
 * Devuelve una representacion del punto como vector de coordenadas
 */
GeoGeograficas.prototype.toVector = function()
{
	return [this.lat, this.lon, this.he];
};



/**

************************************************************************
********
 * Clase GeoCoordPlanas : representa un punto geográfico en coordenaas
planas (UTM, lambert, etc..)

************************************************************************
********
 */


/**
 * Constructor
 * @param {número} x
 * @param {número} y
 * @param {número} he : altura elipsoidal (0, si no se especifica) ;
según el contexto podría ser la altura sobre el geide
 * @param {literal} hs : huso o zona según la proyección (nulo si no
se especifica)
 * @param {número} el: Datum (código interno), por defecto 0 (WGS84)
 */
function GeoCoordPlanas(x, y, he, hs, el)
{
	this.x = x;
	this.y = y;
	this.he = he || 0;
	this.hs = hs || null;
	this.el = el || 0;
};


/**
 * Devuelve una representación del punto como literal
```

```
 */
GeoCoordPlanas.prototype.toString = function()
{
      return "Objeto GeoCoordPlanas (" + this.x + " , " + this.y + " ,
" + this.he + "( zona/huso: " + this.hs + ") , elipsoide:" + this.el +
")";
};


/**
 * Devuelve una representacion del punto como vector de coordenadas
 */
GeoCoordPlanas.prototype.toVector = function()
{
      return [this.x, this.y, this.he];
};




/*
 * Conversión entre geográficas y cartesianas
*/


/**
 * Convierta de coordenadas geográficas a cartesianas
 * @param (GeoGeograficas) : coordenadas geográficas
 * @return (GeoCartesianas): cartesianas en metros
 */
function ELP_geograficasAcartesianas(geograficas)
{
      lat = geograficas.lat;
      lon = geograficas.lon;
      he = geograficas.he;
      el = geograficas.el;

      e2 = (2*ELP_F[el] - 1)/ELP_F[el]/ELP_F[el];
    nu = ELP_A[el] / Math.sqrt(1-e2*Math.sin(lat)*Math.sin(lat));

    x = (nu + he) * Math.cos(lat) * Math.cos(lon);
    y = (nu + he) * Math.cos(lat) * Math.sin(lon);
    z = ((1-e2) * nu + he) * Math.sin(lat);

      return new GeoCartesianas(x, y, z, el);
};



/**
 * Convierte de cartesianas a geográficas
 * @param (GeoCartesianas): coordenadas cartesianas
 * @return (GeoGeograficas): coordenadas geográficas
 */
function ELP_cartesianasAgeograficas(cartesianas)
{
      el = cartesianas.el;
      xc = cartesianas.x;
      yc = cartesianas.y;
      zc = cartesianas.z;
```

```
      //
      // datos iniciales
    lon = Math.atan2(yc, xc);
      e2 = (2*ELP_F[el] - 1)/ELP_F[el]/ELP_F[el];
    aux0 = Math.sqrt(xc*xc+yc*yc);
    lat0 = Math.atan2(zc, aux0*(1-
ELP_A[el]*e2/Math.sqrt(xc*xc+yc*yc+zc*zc))) ;
    lat = lat0;
    err = 1;

      //
      // iteraciones
    while (err>0.0000000000001)
      {
      aux1 = ELP_A[el]*e2*Math.cos(lat0)/ Math.sqrt(1-
e2*Math.sin(lat0)*Math.sin(lat0));
            lat = Math.atan2(zc,aux0-aux1);
            err = lat - lat0 ;
            lat0 = lat;
      }

      he = aux0/Math.cos(lat) - ELP_A[el]/Math.sqrt(1-
e2*Math.sin(lat)*Math.sin(lat));

      return new GeoGeograficas(lat, lon, he, el);
};



/**
 * Convierte de geográficas a UTM
 * @param (GeoGeograficas): coordenadas geograficas
 * @param huso: huso a considerar (si es nulo, se calcula
automaticamente a partir de la longitud)
 * @return (GeoCoordPlanas): coordenadas utm
 */
function ELP_geograficasAutm(geograficas, huso)
{
      el = geograficas.el;
      lat = geograficas.lat;
      lon = geograficas.lon;
      hs = huso;

      if ( !hs)
            hs = detHuso(radAgr(lon));

      e2 = (2*ELP_F[el] - 1)/ELP_F[el]/ELP_F[el];
      c = ELP_A[el]*ELP_F[el]/(ELP_F[el] - 1);
      g2 = (2*ELP_F[el] - 1)/(ELP_F[el]-1)/(ELP_F[el]-1);

      lon0 = (hs - 30.0) * 6 - 3;
      lon0 = grArad(lon0);
      dlon = lon - lon0;

      //
      // ecuaciones de Coticchia-Surace
      ac = Math.cos(lat) * Math.sin(dlon);
      xi = Math.log( (1+ac) / (1-ac) ) / 2;
      eta = Math.atan2( Math.tan(lat), Math.cos(dlon) ) - lat;
      ipsi = ( c / Math.sqrt(1 + g2*Math.cos(lat)*Math.cos(lat) ) ) *
0.9996;
```

```
	zeta = g2 * xi * xi * Math.cos(lat) * Math.cos(lat) / 2;
	a1 = Math.sin(2 * lat);
	a2 = a1 * Math.cos(lat) * Math.cos(lat);
	j2 = lat + a1 / 2;
	j4 = (3 * j2 + a2) / 4;
	j6 = (5 * j4 + a2 * Math.cos(lat) * Math.cos(lat) ) / 3;
	alfa = 3 * g2 / 4;
	beta = 5 * alfa * alfa / 3;
	gamma = 35 * alfa * alfa * alfa /27;
	bfi = 0.9996 * c * (lat - alfa * j2 + beta * j4 - gamma * j6 );

	utmx = xi * ipsi * (1 + zeta / 3) + 500000;
	utmy = eta * ipsi * ( 1 + zeta) + bfi;

	return new GeoCoordPlanas(utmx, utmy, geograficas.he, hs, el);
};



/**
 * Convierte de UTM a geográficas
 * @param (GeoCoordPlanas): coordenada utm (incluyendo elipsoide y
huso)
 * @return (GeoGeograficas): coordenadas geográficas
 */
function ELP_utmAgeograficas(utm)
{
	el = utm.el;
	xutm = utm.x;
	yutm = utm.y;
	huso = utm.hs;

	c = ELP_A[el]*ELP_F[el]/(ELP_F[el] - 1);
	g2 = (2*ELP_F[el] - 1)/(ELP_F[el]-1)/(ELP_F[el]-1);

	x = xutm - 500000;
	y = yutm;

	//
	// determina longitud meridiano central
	lon0 = (huso - 30.0) * 6 - 3;
	lon0 = grArad(lon0);

	//
	// ecuaciones de Coticchia-Surace para el problema inverso
	fi2 = y / (6366197.724 * 0.9996);
	ipsi = ( hay_c / Math.sqrt(1 + g2*Math.cos(fi2)*Math.cos(fi2) )
) * 0.9996;
	ac = x / ipsi;
	a1 = Math.sin(2 * fi2);
	a2 = a1 * Math.cos(fi2) * Math.cos(fi2);
	j2 = fi2 + a1 / 2;
	j4 = (3 * j2 + a2) / 4;
	j6 = (5 * j4 + a2 * Math.cos(fi2) * Math.cos(fi2) ) / 3;
	alfa = 3 * g2 / 4;
	beta = 5 * alfa * alfa / 3;
	gamma = 35 * alfa * alfa * alfa /27;
	bfi = 0.9996 * c * (fi2 - alfa * j2 + beta * j4 - gamma * j6 );
	bc = (y - bfi) / ipsi;
	zeta = g2 * ac * ac * Math.cos(fi2) * Math.cos(fi2) / 2;
	xi = ac * (1 - zeta/3);
```

```
      eta = bc * (1 - zeta) + fi2;
      shx = ( Math.exp(xi) - Math.exp(-xi)) / 2;
      dlon = Math.atan2(shx, Math.cos(eta));
      tau = Math.atan( Math.cos(dlon) * Math.tan(eta) );

      lon = dlon + lon0;
      lat = fi2 + ( 1 + g2 * Math.cos(fi2) * Math.cos(fi2) - 3 * g2 *
Math.sin(fi2) * Math.cos(fi2) * (tau - fi2) / 2) * (tau - fi2);

      return new GeoGeograficas(lat, lon, utm.he, el);
};


/**
 * Converte coordenadas cartesianas entre un elipsoide y WGS84 en los
dos sentidos
 * coordenadas en metros
 * @param (numero): elipsoide
 * @param (GeoCartesianas) : coordenadas cartesianas elipsoide origen
 * @param (numero) :sentido 1 elipsoide a WGS84, -1 WGS84 a elipsoide
 * @return (GeoCartesianas): coordenadas cartesianas elipsoide destino
 */
function ELP_normaliza(el, cartesianas, sentido)
{
      xs = cartesianas.x;
      ys = cartesianas.y;
      zs = cartesianas.z;

      //
      // parámetros de transformación
      dx = sentido*ELP_DX[el];
      dy = sentido*ELP_DY[el];
      dz = sentido*ELP_DZ[el];
      rx = sentido*ELP_RX[el];
      ry = sentido*ELP_RY[el];
      rz = sentido*ELP_RZ[el];
      ds = sentido*ELP_DS[el];

      //
      // convierte parámetros
      rrx = grArad(rx / 3600.0);
      rry = grArad(ry / 3600.0);
      rrz = grArad(rz / 3600.0);
      dds = 1 + ds / 1000000.0;

      //
      // convertir
    xd = dx + dds * (xs + rrz*ys - rry*zs);
    yd = dy + dds * (-rrz*xs + ys + rrx*zs);
    zd = dz + dds * (rry*xs - rrx*ys + zs);

    eld = el;
    if ( sentido > 0 )
      eld = 0;

    return new GeoCartesianas(xd, yd, zd, eld);
};


/**
 * Convierte de coordenadas lambert a coordenadas geograficas
```

```
 * @param (GeoCoordPlanas) : coordenadas lambert (incluyendo la zona
como literal "I", "II", ....)
 * @return (GeoGeograficas) : coordenadas geográficas
 */
function ELP_lambertAgeograficas(lambert)
{
	el = lambert.el;
	xlam = lambert.x;
	ylam = lambert.y;
	zona = lambert.hs;

	var n = 0;
	var C = 0;
	var XS = 0;
	var YS = 0;

	if ( zona == "I")
	{
		n = 0.7604059656;
		C = 11603796.98;
		XS = 600000.000;
		YS = 5657616.674;
	}

	if ( zona == "II")
	{
		n = 0.7289686274;
		C = 11745793.39;
		XS = 600000.000;
		YS = 6199695.768;
	}

	if ( zona == "III")
	{
		n = 0.6959127966;
		C = 11947992.52;
		XS = 600000.000;
		YS = 6791905.085;
	}

	if ( zona == "IV")
	{
		n = 0.6712679322;
		C = 12136281.99;
		XS = 234.358;
		YS = 7239161.542;
	}

	if ( zona == "93" )
	{
		n =  0.7256077650;
		C = 11754255.426;
		XS = 700000.000;
		YS = 12655612.050;
	}

	var lon0 = grArad(8414.025/3600);

	r = Math.sqrt((xlam - XS)*(xlam - XS) + (ylam - YS)*(ylam -
YS));
	g = Math.atan2(xlam - XS, YS - ylam);
```

```
	liso = - Math.log(r/C)/n;

	lon = lon0 + g/n;
	lat = ELP_latisoAlat(liso, el);

	return new GeoGeograficas(lat, lon, lambert.he, el);
};



/**
 * Convierte de coordenadas  geograficas a lambert
 * @param (GeoGeograficas): coordenadas geograficas
 * @param (literal): zona ("I", "II", ... )
 * @return (GeoCoordPlanas): coordenadas lambert
 */
function ELP_geograficasAlambert(geograficas, zona)
{
	el = geograficas.el;
	lat = geograficas.lat;
	lon = geograficas.lon;

	var n = 0;
	var C = 0;
	var XS = 0;
	var YS = 0;

	if ( zona == "I")
	{
		n = 0.7604059656;
		C = 11603796.98;
		XS = 600000.000;
		YS = 5657616.674;
	}

	if ( zona == "II")
	{
		n = 0.7289686274;
		C = 11745793.39;
		XS = 600000.000;
		YS = 6199695.768;
	}

	if ( zona == "III")
	{
		n = 0.6959127966;
		C = 11947992.52;
		XS = 600000.000;
		YS = 6791905.085;
	}

	if ( zona == "IV")
	{
		n = 0.6712679322;
		C = 12136281.99;
		XS = 234.358;
		YS = 7239161.542;
	}

	if ( zona == "93" )
	{
```

```
            n =  0.7256077650;
            C = 11754255.426;
            XS = 700000.000;
            YS = 12655612.050;
      }

      var lon0 = grArad(8414.025/3600);
      e = Math.sqrt((2*ELP_F[el] - 1)/ELP_F[el]/ELP_F[el]);

      liso = Math.log((1+Math.sin(lat))/(1-Math.sin(lat)))/2 - e*
Math.log((1+e*Math.sin(lat))/(1-e*Math.sin(lat))) / 2;
      r = C*Math.exp(-n*liso);
      g = n * (lon - lon0);

      lamx = XS + r*Math.sin(g);
      lamy = YS - r*Math.cos(g);

      return new GeoCoordPlanas(lamx, lamy, geograficas.he, zona, el);
};


/**
 * Determina la latitud a partir de la latitud isométric
 * @param liso: latitud isometrica
 * @param el: codigo de elipsoide (para determinar e)
 * @return: latiud (radianes)
 */
function ELP_latisoAlat(liso, el)
{
      e = Math.sqrt((2*ELP_F[el] - 1)/ELP_F[el]/ELP_F[el]);

      var lt = 2 * Math.atan(Math.exp(liso)) - Math.PI / 2;

      err = 1;

      //
      // iteraciones
    while (err>0.0000000000001)
      {
      al = Math.pow((1+e*Math.sin(lt))/(1-e*Math.sin(lt)),e/2);
      bl = al * Math.exp(liso);
      lti = 2 * Math.atan(bl) - Math.PI / 2;
      err = Math.abs(lt-lti);
      lt = lti;
      }

    return lt;
};
```

ANEXO 2. MODELO DIGITAL DE ELEVACIONES

```
/**

*****************************************************************************
********
 * Clase GeoMdeGlobal : modelo digital de elevaciones de lectura
completa

*****************************************************************************
********
 */


/**
 * Constructor
 * @param {GeoMdeGlobalOpciones} opc: parámetros de difinición del mde
 */

function GeoMdeGlobal(opc)
{
      this.url          = opc.url;
      this.src          = opc.src;
      this.paso         = opc.paso;
      this.origenX      = opc.origenX;
      this.origenY      = opc.origenY;
      this.filas        = opc.filas;
      this.columnas     = opc.columnas;

      this.grid = null;

      var req = new GeoAjax();
      req.Init();

      if ( req.request )
      {
            var self = this;
            req.request.onreadystatechange =
function(){self._mdeLeido(req);};
            req.EnviaDatosGet(this.url);
      }
}



/**
 * Determina la altura de un punto
 * @param {GeoPunto} punto: punto cuya altura se desea conocer
(devuelve null si el punto se sale de la malla)
 * @param {Literal} scoor: sistema de coordenadas al que está referido
el punto anterior
 */
GeoMdeGlobal.prototype.getAltura = function(punto, scoor)
{
      x = punto.x;
      y = punto.y;

      if ( this.src != scoor)
      {
            cor = transforma(scoor, this.src, [punto.x , punto.y]);
```

```
            x = parseFloat(cor[0]);
            y = parseFloat(cor[1]);
      }

      c0 = Math.floor((x-this.origenX)/this.paso);
      f0 = Math.floor((this.origenY-y)/this.paso);

      dx = x - c0*this.paso - this.origenX;
      dy = this.origenY - f0*this.paso - y;

      if ( c0 < 0 || c0 > this.columnas-2 )
            return null;

      if ( f0 < 0 || f0 > this.filas-2 )
            return null;

      pos = f0 * this.columnas + c0;
      h00 = parseFloat(this.grid[pos]);
      h01 = parseFloat(this.grid[pos+1]);
      h10 = parseFloat(this.grid[pos+this.columnas]);
      h11 = parseFloat(this.grid[pos+this.columnas+1]);

      hf0 = h00 + dx*(h01-h00)/this.paso;
      hf1 = h10 + dx*(h11-h10)/this.paso;

      return (hf0 + dy * (hf1-hf0)/this.paso);
};



GeoMdeGlobal.prototype._mdeLeido = function(req)
{
      error = req.Respuesta();
      if (!error)
      {
            var respuesta = req.request.responseText;
            if (respuesta != null)
            {
                  this.grid = respuesta.split(',');
            }
      }
};
```

ANEXO 3. GESTIÓN DEL GPS

3.1. Gestión de las tramas NMEA

```
package es.rsbsistema.gps;

import java.util.Date;


public class TareaGps
{
	protected boolean fin = false;
	protected String error = "No se están leyendo datos del GPS";

	private int gpsfix = 0;
	private int gpscalidad = 0;
	private String gpsutc = null;
	private double gpslon = 0;
	private double gpslat = 0;
	private double gpsvelocidad = 0;
	private double gpstrkang = 0;
	private String gpsfecha = null;
	private double gpshorto = 0;
	private double gpshgeoide = 0;

	private double gpsVh = 0;
	private double[] gpsdate;
	private double[] gpsaltura;

	private String[] historia = null;

	public void init()
	{
		historia = new String[10];
		for (int i=0; i<10 ; i++ )
			historia[i] = "";

		gpsdate = new double[2];
		gpsaltura = new double[2];
		for (int i=0 ; i<2 ; i++)
		{
			gpsdate[i] = new Date().getTime();
			gpsaltura[i] = 0;
		}
	}

	public void procesaTrama(String datos)
	{
		for (int i=0 ; i<9 ; i++)
			historia[i] = historia[i+1];
		historia[9] = datos;

		String[] elem = datos.split(",");
		if (elem.length > 0 )
		{
			if ( elem[0].equals("$GPGSA") )
			{
				error = null;
				if ( elem.length > 2 )
					gpsfix = Integer.parseInt(elem[2]);
```

```
                  }

                  if ( elem[0].equals("$GPGGA") )
                  {
                        error = null;
                        if ( elem.length > 6 )
                        {
                              gpsutc = elem[1].substring(0, 2) + ":" +
elem[1].substring(2, 4) + ":" + elem[1].substring(4, 6);

                              gpslat = getLatLon(elem[2]);
                              if ( !elem[3].equals("N") )
                                    gpslat = -gpslat;

                              gpslon = getLatLon(elem[4]);
                              if ( !elem[5].equals("E") )
                                    gpslon = -gpslon;

                              gpscalidad = Integer.parseInt(elem[6]);
                              gpshorto = Double.parseDouble(elem[9]);
                              gpshgeoide =
Double.parseDouble(elem[11]);

                              gpsdate[1] = gpsdate[0];
                              gpsaltura[1] = gpsaltura[0];
                              gpsdate[0] = new Date().getTime();
                              gpsaltura[0] = gpshorto;
                        }
                  }

                  if ( elem[0].equals("$GPRMC") )
                  {
                        error = null;
                        if ( elem.length > 9 )
                        {
                              gpsutc = elem[1].substring(0, 2) + ":" +
elem[1].substring(2, 4) + ":" + elem[1].substring(4, 6);

                              if ( elem[2].equals("A") )
                              {
                                    gpslat = getLatLon(elem[3]);
                                    if ( !elem[4].equals("N") )
                                          gpslat = -gpslat;

                                    gpslon = getLatLon(elem[5]);
                                    if ( !elem[6].equals("E") )
                                          gpslon = -gpslon;

                                    gpsvelocidad =
Double.parseDouble(elem[7]);
                                    gpstrkang =
Double.parseDouble(elem[8]);
                                    gpsfecha = elem[9].substring(0, 2)
+ "/" + elem[9].substring(2, 4) + "/" + elem[9].substring(4, 6);
                              }
                        }
                  }
            }
      }

      public double[] getPosicion()
```

```
	{
		double[] ret = null;

		if ( gpsfix > 1 && gpscalidad > 0 )
		{
			ret = new double[2];
			ret[0] = gpslon;
			ret[1] = gpslat;
		}

		return ret;
	}

	public double[] getAltura()
	{
		double[] ret = null;

		if ( gpsfix > 2 && gpscalidad > 0 )
		{
			ret = new double[2];
			ret[0] = gpshorto;
			ret[1] = gpshgeoide;
		}

		return ret;
	}

	public double[] getVelocidad()
	{
		double[] ret = null;

		if ( gpsfix > 1 && gpscalidad > 0 )
		{
			gpsVh = 1000 * (gpsaltura[1] - gpsaltura[0] ) /
(gpsdate[1] - gpsdate[0]);

			ret = new double[3];
			ret[0] = gpsvelocidad;
			ret[1] = gpstrkang;
			ret[2] = gpsVh;
		}

		return null;
	}


	public String getGpsutc() {
		return gpsutc;
	}

	public String getGpsfecha() {
		return gpsfecha;
	}


	public String getError()
	{
		return error;
	}

	public void senylaFin()
```

```
	{
		fin = true;
	}

	public double getLatLon(String data)
	{
		double ret = 0;

		int coma = data.indexOf(".");
		if ( coma > 2 )
		{
			int gr = Integer.parseInt(data.substring(0, coma -2
));
			double mi = Double.parseDouble(data.substring(coma-
2));
			ret = mi / 60.0 + gr;
		}

		return ret;
	}

	public String getInfo()
	{
		String ret = "";
		ret += "<?xml version='1.0' encoding='ISO-8859-1' ?>";
		ret += "<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0
Transitional//EN' 'http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd'>";
		ret += "<html xmlns='http://www.w3.org/1999/xhtml'>";

		ret += "<head>";
		ret += "<meta http-equiv='Content-Type' content='text/html;
charset=ISO-8859-1' />";
		ret += "<title>Información sobre el GPS</title>";
		ret += "</head>";

		ret += "<body>";

		ret += "<h2>Últimas tramas recibidas</h2>";
		ret += "<table><tbody>";

		for (int i=0 ; i<10 ; i++ )
			if ( historia[i] != null && historia[i].length() > 0
)
			{
				ret += "<tr><td>";
				ret += historia[i];
				ret += "</td></tr>";
			}

		ret += "</tbody></table>";
		ret += "</body>";
		ret += "</html>";

		return ret;
	}

}
```

3.2. Servlet de intercomunicación de la posición actual

```
package es.rsbsistema.smap.servlet;

import ipsige.entorno.Configuracion;

import java.io.IOException;

import javax.servlet.Servlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import scfa.parametros.ScfaConfiguracion;

import es.rsbsistema.gps.TareaGps;
import es.rsbsistema.gps.TareaGpsCOM;
import es.rsbsistema.gps.TareaGpsUDP;


/**
 * Servlet implementation class GPSServlet
 */
public class GPSServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;
	private static TareaGps tgps = null;

   /**
    * @see HttpServlet#HttpServlet()
    */
   public GPSServlet()
   {
      super();
   }


	/**
	 * @see Servlet#init(ServletConfig)
	 */
	public void init(ServletConfig config) throws ServletException
	{
		super.init(config);

		ScfaConfiguracion scfaconf = new ScfaConfiguracion();
		String server = scfaconf.getServerConf();

		String fileConf = null;
```

```
                fileConf = config.getInitParameter("ipSigeConfiguracion");
                Configuracion.setFichero(server + fileConf);
        }


        /**
         * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
response)
         */
        protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
        {
                serviceRequest(request, response);
        }

        /**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
response)
         */
        protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
        {
                serviceRequest(request, response);
        }


        protected void serviceRequest(HttpServletRequest request, HttpServletResponse
response)      throws ServletException, IOException
        {
                String proces = request.getParameter("PROCESO");

                if ( proces != null && proces.equals("INICIO") )
                {
                        if ( tgps == null )
                        {
                                HttpSession session = request.getSession();

                                int puerto = 0;
                                Configuracion conf = Configuracion.Registra(session);
                                if ( conf != null )
                                        puerto = conf.GetParametroInt("gps.udp.puerto");

                                if ( puerto != 0 )
                                {
                                        tgps = new TareaGpsUDP();
                                        ((TareaGpsUDP)tgps).setPuerto(puerto);
                                }
                                else
                                {
                                        String comprt = "";
```

```
                                String baudRate = "";
                                String byteSize = "";
                                String stopBits = "";
                                String parity = "";

                                    if ( conf != null )
                                    {
                                            comprt =
conf.GetParametro("gps.com.puerto");
                                            baudRate =
conf.GetParametro("gps.com.baudRate");
                                            byteSize =
conf.GetParametro("gps.com.byteSize");
                                            stopBits =
conf.GetParametro("gps.com.stopBits");
                                            parity =
conf.GetParametro("gps.com.parity");

                                            if ( comprt != null && comprt.length() > 0 )
                                            {
                                                    tgps = new TareaGpsCOM();

        ((TareaGpsCOM)tgps).setPuerto(comprt);

        ((TareaGpsCOM)tgps).setBaudRate(baudRate);

        ((TareaGpsCOM)tgps).setByteSize(byteSize);

        ((TareaGpsCOM)tgps).setStopBits(stopBits);

        ((TareaGpsCOM)tgps).setParity(parity);
                                            }
                                    }
                            }

                            if ( tgps != null )
                                    new Thread((Runnable)tgps).start();
                    }
            }

            if ( proces != null && proces.equals("POSICION"))
            {
                    String xml = "<?xml version='1.0' encoding='utf-8'?>";
                    xml += "<feed xmlns=\"http://www.w3.org/2005/Atom\"
xmlns:georss=\"http://www.georss.org/georss\"
xmlns:gml=\"http://www.opengis.net/gml\">";
                    xml += "<author>ipSigeMap</author>";

                    String error = "No se ha iniciado sistema";
```

```
                    if ( tgps != null )
                    {
                        error = tgps.getError();
                        if ( error == null )
                        {
                            double[] alt = tgps.getAltura();
                            double[] pos = tgps.getPosicion();
                          double[] vel = tgps.getVelocidad();

                            if ( pos == null || pos.length < 2 )
                                error = "No hay posición válida";
                            else
                            {
                                String lat = "LAT: ";
                                if (pos[1] >= 0 )
                                    lat += String.valueOf(pos[1]) + "
N";
                                else
                                    lat += String.valueOf(-pos[1]) + "
S";

                                String lon = "LON: ";
                                if (pos[0] >= 0 )
                                    lon += String.valueOf(pos[0]) + "
E";
                                else
                                    lon += String.valueOf(-pos[0]) + "
W";

                                xml += "<entry>";
                                xml += "<id>Mi posicion</id>";
                                xml += "<category></category>";
                                xml += "<summary>" + lat + " , " + lon;

                                if ( alt != null )
                                    xml += " , H: " + String.valueOf(alt[0]);

                                xml += "</summary>";
                                xml += "<updated>" + tgps.getGpsfecha() + " "
+ tgps.getGpsutc() + "</updated>";

                                xml += "<georss:where>";
                                xml += "<gml:Point>";
                                xml += "<gml:pos>" + String.valueOf(pos[0]) +
" " + String.valueOf(pos[1]);

                                if ( alt != null )
                                    xml += " " + String.valueOf(alt[0]);

                                xml += "</gml:pos>";
```

```
                                        xml += "</gml:Point>";

                                        if ( vel != null )
                                        {
                                                xml += "<velocidad>" +
String.valueOf(vel[0])+ " " + String.valueOf(vel[1]) + " " + String.valueOf(vel[2]);
                                                xml += "</velocidad>";
                                        }

                                        xml += "</georss:where>";

                                        xml += "</entry>";
                                }
                        }
                }

                if ( error != null )
                        xml += "<category term='error' label='" + error + "'/>";


                xml += "</feed>";

                response.setContentType("text/xml; charset=UTF-8");
                response.setHeader("Cache-Control", "no-cache");
                response.getWriter().write(xml);
        }


        if ( proces != null && proces.equals("INFO"))
        {
                String xml = "";
                if ( tgps != null )
                        xml = tgps.getInfo();

                response.setContentType("text/xml; charset=UTF-8");
                response.setHeader("Cache-Control", "no-cache");
                response.getWriter().write(xml);
        }
    }

    public void destroy()
    {
        if ( tgps != null )
                tgps.senylaFin();

        super.destroy();
    }

}
```

ANEXO 4. AMPLIACIÓN DE LA INTERFAZ DE USUARIO DEL VISOR

4.1. Visualización de las coordenadas

```
GeoIpSige.prototype.activaCoordenadas = function(action)
{
	if ( action )
	{
	if ( !this.listenerMouseMove )
	{
		var self = this;
		this.listenerMouseMove = GeoEventos.addEvent(this.map,
"mousemove", function(point)
			{
				txt =
self.map.sisact.getTextoCoordenadas(point);
				if (self.mdeGlobal)
				{
					h = self.mdeGlobal.getAltura(point,
self.map.sisact.src);
					if (h)
						txt += "    h = " +
self.mdeGlobal.getAltura(point, self.map.sisact.src);
				}

	document.getElementById("coordenadas").innerHTML = txt;
			});
	}
	}
	else
	{
		GeoEventos.removeEvent(this.listenerMouseMove);
	this.listenerMouseMove = null;
	}
};
```

4.2. Control helicóptero

```
	function GeoControlHelicoptero()
	{
	  this.mapa = null;
	  this.control = null;

		this.heli = null;
		this.on = false;
	}


	GeoControlHelicoptero.prototype.initialize = function(map)
	{
		this.mapa = map;
		var self = this;

		// ????
		imagen1=new Image;
		imagen1.src="Imagenes/luzroja.gif";
```

```
                  imagen2=new Image;
                  imagen2.src="Imagenes/luznegra.gif";
                  imagen3=new Image;
                  imagen3.src="Imagenes/pausa.png";
                  imagen4=new Image;
                  imagen4.src="Imagenes/play.png";

                  if ( geo.posicion )
                  {
                        geo.posicion.inicio();
                  GeoEventos.addEvent(geo.posicion, "posicion",
function(posicion)
                  {
                        if ( posicion.x && posicion.y )
                        {
                              var h = 0;
                              if ( geo.mdeGlobal )
                                          h =
geo.mdeGlobal.getAltura(new GeoPunto(posicion.x, posicion.y),
"EPSG:4326");

                                    if ( !h)
                                          h = 0;

                                    if ( self.heli )

      self.mapa.eliminaSuperposicion(self.heli);

                                    // determina esquinas del
helicoptero en UTM
                              var cor = transforma("EPSG:4326",
"EPSG:23030", [posicion.x, posicion.y]);

                              var x0 = cor[0];
                              var y0 = cor[1] - 30;

                              var x1 = cor[0];
                              var y1 = cor[1] + 30;

                              var x2 = cor[0] + 30 + posicion.velhor *
4;
                              var y2 = cor[1] + 30;

                              var x3 = cor[0] + 30 + posicion.velhor *
4;
                              var y3 = cor[1] -30;

                              // convierte al sistema utilizado
                              cor = transforma("EPSG:23030",
geo.sisini.src, [x0, y0]);
                              var p0 = new GeoPunto(cor[0], cor[1]);

                              cor = transforma("EPSG:23030",
geo.sisini.src, [x1, y1]);
                              var p1 = new GeoPunto(cor[0], cor[1]);

                              cor = transforma("EPSG:23030",
geo.sisini.src, [x2, y2]);
                              var p2 = new GeoPunto(cor[0], cor[1]);
```

```
                                cor = transforma("EPSG:23030",
geo.sisini.src, [x3, y3]);
                                var p3 = new GeoPunto(cor[0], cor[1]);

                                // construye helicóptero
                                      self.heli = new
Geo3DParalepipedo('heli', [p0, p1, p2, p3], posicion.z - 10,
posicion.z, "#f00", 1);

      self.mapa.addSuperposicion(self.heli);

                                      // acctualiza campos del control
                                      var obj =
document.getElementById('helx');
                                      obj.value = x0;

                                      obj =
document.getElementById('hely');
                                      obj.value = y0;

                                      obj =
document.getElementById('helz');
                                      obj.value = posicion.z;

                                      obj =
document.getElementById('helv');
                                      obj.value = posicion.velhor;

                                      // control choque
//                                    if (
cole.interseccionParalepipedoGen(self.heli) )
//                                          alert('CHOCA');

                                      if( self.on )
                                      {
                                            if (
cole.interseccionParalepipedoGen(self.heli) )
                                            {

      document.images['AlarmaObstaculos'].src=imagen1.src;
                                            }
                                            else

      document.images['AlarmaObstaculos'].src=imagen2.src;

                                            //control altura
                                            var proteccion = 10;
                                            if ( posicion.z <
h+proteccion )
                                            {

      document.images['AlarmaSuelo'].src=imagen1.src;
                                            }
                                            else

      document.images['AlarmaSuelo'].src=imagen2.src;
                                      }
                        }
                  });
                  }
```

```
                obj = document.createElement('div');
//              obj.style.width = '200px';
//              obj.style.height = '100px';

                var html = "<table style='font-
size:12px;'><tbody><tr><td><img src='Imagenes/play.png' name='ON/OFF'
id='onoff'/></td><td></td></tr>";
                html += "<tr><th>Posición</th><th></th></tr>";
                html += "<tr><td><input id='helx' type='text'
readonly='readonly'/></td><td><input id='hely' type='text'
readonly='readonly'/></td></tr>";
                html += "<tr><th>Altura</th><th>Velocidad</th></tr>";
                html += "<tr><td><input id='helz' type='text'
readonly='readonly'/></td><td><input id='helv' type='text'
readonly='readonly'/></td></tr>";
                html += "<tr><th>Alarma suelo</th><th>Alarma
obstaculo</th></tr>";
                html += "<th><img src='Imagenes/luznegra.gif'
name='AlarmaSuelo'/></th><th><img src='Imagenes/luznegra.gif'
name='AlarmaObstaculos'/></th></tbody></table>";

                obj.innerHTML = html;

                this.control = new GeoVentana(new GeoPixel(50,
getAltoVentana() - 350), obj, this.mapa);

                var onoff = document.getElementById('onoff');
                onoff.onclick = function(){self.encender()};
                onoff.style.cursor = "pointer";
        }


        GeoControlHelicoptero.prototype.finaliza = function(map)
        {
              if ( this.heli)
                      this.mapa.eliminaSuperposicion(this.heli);

        }

        GeoControlHelicoptero.prototype.encender = function ()
          {
                if (this.on == false)
                {
                      document.images['ON/OFF'].src=imagen3.src;
                      this.on = true;
                }
                else
                {
                      document.images['ON/OFF'].src=imagen4.src;
                      this.on = false;
                }
          }
```

4.3. Control de la posición

```
        function GeoControlPosicion()
        {
          this.mapa = null;
          this.control = null;
```

```
            this.posicion = new GeoPosicion();


                  // posicion inicio del control posición
                  this.x0 = 174206;
                  this.y0 = 4130235;

                  this.delta = 50;
                  this.altura = 0;
            }


            GeoControlPosicion.prototype.initialize = function(map)
            {
                  this.mapa = map;


                  // crea control de posición
                  var self = this;

              this.control = document.createElement('div');
              this.control.style.position = 'absolute';
              this.control.style.left = '50px';
              this.control.style.top = (getAltoVentana() - 150) + 'px';
              this.control.style.zIndex = '5';

              var html = "<div id='activacion' style='font-size: 12px;
font-weight: bold; background-color:
transparent;'><table><tbody><tr><td><input type='button' value='B'
id='bajar'/></td><td><input type='button' value='+'
id='arriba'/></td><td><input type='button' value='S'
id='subir'/></td></tr><tr><td><input type='button' value='+'
id='izquierda'/></td><td></td><td><input type='button' value='+'
id='derecha'/></td></tr><tr><td><input type='button' value='<<'
id='mitad'/></td><td><input type='button' value='+'
id='abajo'/></td><td><input type='button' value='>>'
id='doble'/></td></tr></tbody></table></div>";
              this.control.innerHTML = html;

              var cont = map.getContenedor(GEO_CONTROLES);

              if (cont)
                    cont.appendChild(this.control);

              var obj = document.getElementById('arriba');
              obj.onclick = function(){self.arriba();};
              obj.ondblclick =
function(event){GeoEventos.evitaEventos(event);};

              obj = document.getElementById('izquierda');
              obj.onclick = function(){self.izquierda();};
              obj.ondblclick =
function(event){GeoEventos.evitaEventos(event);};

              obj = document.getElementById('derecha');
              obj.onclick = function(){self.derecha();};
              obj.ondblclick =
function(event){GeoEventos.evitaEventos(event);};

              obj = document.getElementById('abajo');
```

```
            obj.onclick = function(){self.abajo();};
            obj.ondblclick =
function(event){GeoEventos.evitaEventos(event);};

            obj = document.getElementById('mitad');
            obj.onclick = function(){self.mitad();};
            obj.ondblclick =
function(event){GeoEventos.evitaEventos(event);};

            obj = document.getElementById('doble');
            obj.onclick = function(){self.doble();};
            obj.ondblclick =
function(event){GeoEventos.evitaEventos(event);};

            obj = document.getElementById('bajar');
            obj.onclick = function(){self.bajar();};
            obj.ondblclick =
function(event){GeoEventos.evitaEventos(event);};

            obj = document.getElementById('subir');
            obj.onclick = function(){self.subir();};
            obj.ondblclick =
function(event){GeoEventos.evitaEventos(event);};
          }


          GeoControlPosicion.prototype.finaliza = function(map)
          {
                  // eliminar control de posicións
                  var cont = map.getContenedor(GEO_CONTROLES);
            if (cont)
                  cont.removeChild(this.control);

            this.control = null;
          }


          GeoControlPosicion.prototype.doble = function()
          {
            this.delta *= 2;
                  this.simula();
          }

          GeoControlPosicion.prototype.mitad = function()
          {
                this.delta /= 2;
                  this.simula();
          }

          GeoControlPosicion.prototype.arriba = function()
            {
                  this.y0 += this.delta;
                  this.simula();
            }
          GeoControlPosicion.prototype.izquierda = function()
            {
                  this.x0 -= this.delta;
                  this.simula();
            }
          GeoControlPosicion.prototype.derecha = function()
            {
```

```
                this.x0 += this.delta;
                this.simula();
          }
        GeoControlPosicion.prototype.abajo = function()
          {
                this.y0 -= this.delta;
                this.simula();
          }

        GeoControlPosicion.prototype.bajar = function()
          {
                this.altura -= 10;
                this.simula();
          }
        GeoControlPosicion.prototype.subir = function()
          {
                this.altura += 10;
                this.simula();
          }


        GeoControlPosicion.prototype.simula = function()
        {
                var cor = transforma("EPSG:23030", "EPSG:4326",
[this.x0, this.y0]);

                this.posicion.error = null;
                this.posicion.x = cor[0];
                this.posicion.y = cor[1];
                this.posicion.z = this.altura;
                this.posicion.velhor = this.delta;
                this.posicion.rumbo = 0;
                this.posicion.velver = 0;
                this.posicion.hint = "Posición simulada";
                this.posicion.fecha = "";

                if ( geo.posicion )
                      geo.posicion.simulacion(this.posicion);
        }
```